

# VxWorks® OS Libraries

# 5.5

API REFERENCE



---

Copyright © 2002 Wind River Systems, Inc.

ALL RIGHTS RESERVED. No part of this publication may be copied in any form, by photocopy, microfilm, retrieval system, or by any other means now known or hereafter invented without the prior written permission of Wind River Systems, Inc.

AutoCode, Embedded Internet, Epilogue, ESp, FastJ, IxWorks, MATRIX<sub>X</sub>, pRISM, pRISM+, pSOS, RouterWare, Tornado, VxWorks, *wind*, WindNavigator, Wind River Systems, WinRouter, and Xmath are registered trademarks or service marks of Wind River Systems, Inc. or its subsidiaries.

Attaché Plus, BetterState, Doctor Design, Embedded Desktop, Emissary, Envoy, How Smart Things Think, HTMLWorks, MotorWorks, OSEKWorks, Personal JWorks, pSOS+, pSOSim, pSOSystem, SingleStep, SNiFF+, VSPWorks, VxDCOM, VxFusion, VxMP, VxSim, VxVMI, Wind Foundation Classes, WindC++, WindManage, WindNet, Wind River, WindSurf, and WindView are trademarks or service marks of Wind River Systems, Inc. or its subsidiaries. This is a partial list. For a complete list of Wind River trademarks and service marks, see the following URL:

<http://www.windriver.com/corporate/html/trademark.html>

Use of the above marks without the express written permission of Wind River Systems, Inc. is prohibited. All other trademarks, registered trademarks, or service marks mentioned herein are the property of their respective owners.

#### **Corporate Headquarters**

Wind River Systems, Inc.  
500 Wind River Way  
Alameda, CA 94501-1153  
U.S.A.

toll free (U.S.): 800/545-WIND  
telephone: 510/748-4100  
facsimile: 510/749-2010

For additional contact information, please visit the Wind River URL:

<http://www.windriver.com>

For information on how to contact Customer Support, please visit the following URL:

<http://www.windriver.com/support>

# Contents

## **1: Libraries**

This volume provides reference entries for VxWorks OS libraries, arranged alphabetically. Each entry lists the routines found in the library, including a one-line synopsis of each and a general description of their use.

Individual reference entries for each of the available functions in these libraries is provided in section 2.

## **2: Routines**

This section provides reference entries for each of the routines found in the VxWorks OS libraries documented in section 1.

## **Keyword Index**

This section is a “permuted index” of keywords found in the NAME line of each reference entry. The keyword for each index item is left-aligned in column 2. The remaining words in column 1 and 2 show the context for the keyword.



# 1

## Libraries

<b>aioPxLib</b>	– asynchronous I/O (AIO) library (POSIX).....	9
<b>aioPxShow</b>	– asynchronous I/O (AIO) show library.....	13
<b>aioSysDrv</b>	– AIO system driver .....	13
<b>ansiAssert</b>	– ANSI assert documentation.....	13
<b>ansiCtype</b>	– ANSI ctype documentation.....	14
<b>ansiLocale</b>	– ANSI locale documentation .....	15
<b>ansiMath</b>	– ANSI math documentation .....	15
<b>ansiSetjmp</b>	– ANSI setjmp documentation.....	16
<b>ansiStdarg</b>	– ANSI stdarg documentation .....	17
<b>ansiStdio</b>	– ANSI stdio documentation.....	18
<b>ansiStdlib</b>	– ANSI stdlib documentation .....	22
<b>ansiString</b>	– ANSI string documentation .....	24
<b>ansiTime</b>	– ANSI time documentation .....	25
<b>arpLib</b>	– Address Resolution Protocol (ARP) table manipulation library .....	26
<b>bLib</b>	– buffer manipulation library .....	28
<b>bootConfig</b>	– system configuration module for boot ROMs.....	29
<b>bootInit</b>	– ROM initialization module .....	30
<b>bootLib</b>	– boot ROM subroutine library .....	31
<b>bootpLib</b>	– Bootstrap Protocol (BOOTP) client library .....	33
<b>bpfdrv</b>	– Berkeley Packet Filter (BPF) I/O driver library .....	35
<b>cache4kcLib</b>	– MIPS 4kc cache management library.....	37
<b>cacheArchLib</b>	– architecture-specific cache management library.....	37
<b>cacheAuLib</b>	– Alchemy Au cache management library.....	38
<b>cacheLib</b>	– cache management library .....	38
<b>cacheR3kLib</b>	– MIPS R3000 cache management library .....	47
<b>cacheR4kLib</b>	– MIPS R4000 cache management library .....	47
<b>cacheR5kLib</b>	– MIPS R5000 cache management library .....	48
<b>cacheR7kLib</b>	– MIPS R7000 cache management library .....	48
<b>cacheR10kLib</b>	– MIPS R10000 cache management library.....	49
<b>cacheR32kLib</b>	– MIPS RC32364 cache management library .....	49

<b>cacheR33kLib</b>	- MIPS R33000 cache management library .....	50
<b>cacheR333x0Lib</b>	- MIPS R333x0 cache management library .....	50
<b>cacheSh7040Lib</b>	- Hitachi SH7040 cache management library .....	51
<b>cacheSh7604Lib</b>	- Hitachi SH7604/SH7615 cache management library .....	51
<b>cacheSh7622Lib</b>	- SH7622 cache management library .....	52
<b>cacheSh7700Lib</b>	- Hitachi SH7700 cache management library .....	52
<b>cacheSh7729Lib</b>	- Hitachi SH7729 cache management library .....	53
<b>cacheSh7750Lib</b>	- Hitachi SH7750 cache management library .....	53
<b>cacheSun4Lib</b>	- Sun-4 cache management library .....	54
<b>cacheTx49Lib</b>	- Toshiba Tx49 cache management library .....	54
<b>cbioLib</b>	- cached block I/O library .....	55
<b>cdromFsLib</b>	- ISO 9660 CD-ROM read-only file system library .....	59
<b>clockLib</b>	- clock library (POSIX) .....	61
<b>cplusLib</b>	- basic run-time support for C++ .....	62
<b>dbgArchLib</b>	- architecture-dependent debugger library .....	63
<b>dbgLib</b>	- debugging facilities .....	65
<b>dcacheCbio</b>	- disk cache driver .....	67
<b>dhcpcBootLib</b>	- DHCP boot-time client library .....	71
<b>dhcpcCommonLib</b>	- DHCP client interface shared code library .....	72
<b>dhcpcLib</b>	- Dynamic Host Configuration Protocol (DHCP) run-time client API .....	73
<b>dhcpcShow</b>	- DHCP run-time client information display routines .....	75
<b>dhcprLib</b>	- DHCP relay agent library .....	75
<b>dhcpsLib</b>	- Dynamic Host Configuration Protocol (DHCP) server library .....	76
<b>dirLib</b>	- directory handling library (POSIX) .....	81
<b>distIfShow</b>	- distributed objects interface adapter show routines (VxFusion) .....	83
<b>distLib</b>	- distributed objects initialization and control library (VxFusion) .....	83
<b>distNameLib</b>	- distributed name database library (VxFusion) .....	84
<b>distNameShow</b>	- distributed name database show routines (VxFusion) .....	85
<b>distTBufLib</b>	- distributed objects telegram buffer library (VxFusion) .....	85
<b>dosFsFmtLib</b>	- MS-DOS media-compatible file system formatting library .....	86
<b>dosFsLib</b>	- MS-DOS media-compatible file system library .....	86
<b>dpartCbio</b>	- generic disk partition manager .....	98
<b>dspLib</b>	- dsp support library .....	100
<b>dspShow</b>	- dsp show routines .....	100
<b>envLib</b>	- environment variable library .....	101
<b>errnoLib</b>	- error status library .....	102
<b>etherMultiLib</b>	- a library to handle Ethernet multicast addresses .....	104
<b>eventLib</b>	- VxWorks events library .....	104
<b>excArchLib</b>	- architecture-specific exception-handling facilities .....	105
<b>excLib</b>	- generic exception handling facilities .....	106
<b>fiolib</b>	- formatted I/O library .....	108
<b>floatLib</b>	- floating-point formatting and scanning library .....	109
<b>fppArchLib</b>	- architecture-dependent floating-point coprocessor support .....	109
<b>fppLib</b>	- floating-point coprocessor support library .....	112
<b>fppShow</b>	- floating-point show routines .....	113

<b>ftpdLib</b>	– File Transfer Protocol (FTP) server .....	113
<b>ftpLib</b>	– File Transfer Protocol (FTP) library .....	115
<b>ftruncate</b>	– POSIX file truncation .....	117
<b>hostLib</b>	– host table subroutine library .....	118
<b>icmpShow</b>	– ICMP Information display routines .....	119
<b>ifIndexLib</b>	– interface index library .....	119
<b>ifLib</b>	– network interface library .....	120
<b>igmpShow</b>	– IGMP information display routines .....	121
<b>inetLib</b>	– internet address manipulation routines .....	121
<b>inflateLib</b>	– inflate code using public domain <b>zlib</b> functions .....	123
<b>intArchLib</b>	– architecture-dependent interrupt library .....	123
<b>intLib</b>	– architecture-independent interrupt subroutine library .....	125
<b>ioLib</b>	– I/O interface library .....	125
<b>iosLib</b>	– I/O system library .....	127
<b>iosShow</b>	– I/O system show routines .....	127
<b>ipFilterLib</b>	– IP filter hooks library .....	128
<b>ipProto</b>	– an interface between the BSD IP protocol and the MUX .....	128
<b>kernelLib</b>	– VxWorks kernel library .....	129
<b>ledLib</b>	– line-editing library .....	131
<b>loadLib</b>	– object module loader .....	133
<b>loginLib</b>	– user login/password subroutine library .....	134
<b>logLib</b>	– message logging library .....	136
<b>lstLib</b>	– doubly linked list subroutine library .....	137
<b>m2IcmpLib</b>	– MIB-II ICMP-group API for SNMP Agents .....	139
<b>m2IfLib</b>	– MIB-II interface-group API for SNMP agents .....	139
<b>m2Igmp</b>	– helper file for igmp Mib .....	141
<b>m2IpLib</b>	– MIB-II IP-group API for SNMP agents .....	142
<b>m2Lib</b>	– MIB-II API library for SNMP agents .....	144
<b>m2RipLib</b>	– VxWorks interface routines to RIP for SNMP Agent .....	147
<b>m2SysLib</b>	– MIB-II system-group API for SNMP agents .....	148
<b>m2TcpLib</b>	– MIB-II TCP-group API for SNMP agents .....	150
<b>m2UdpLib</b>	– MIB-II UDP-group API for SNMP agents .....	152
<b>mathALib</b>	– C interface library to high-level math functions .....	153
<b>mathHardLib</b>	– hardware floating-point math library .....	155
<b>mathSoftLib</b>	– high-level floating-point emulation library .....	155
<b>memDrv</b>	– pseudo-memory device driver .....	156
<b>memLib</b>	– full-featured memory partition manager .....	158
<b>memPartLib</b>	– core memory partition manager .....	160
<b>memShow</b>	– memory show routines .....	161
<b>mmanPxLib</b>	– memory management library (POSIX) .....	162
<b>mmuMapLib</b>	– MMU mapping library for ARM Ltd. processors .....	162
<b>mmuPro32Lib</b>	– MMU library for PentiumPro/2/3/4 32 bit mode .....	163
<b>mmuSh7700Lib</b>	– Hitachi SH7700 MMU support library .....	168
<b>mmuSh7750Lib</b>	– Hitachi SH7750 MMU support library .....	172
<b>moduleLib</b>	– object module management library .....	176

<b>mountLib</b>	– mount protocol library .....	178
<b>mqPxLib</b>	– message queue library (POSIX).....	179
<b>mqPxShow</b>	– POSIX message queue show.....	180
<b>msgQDistGrpLib</b>	– distributed message queue group library (VxFusion).....	180
<b>msgQDistGrpShow</b>	– distributed message queue group show routines (VxFusion).....	181
<b>msgQDistLib</b>	– distributed objects message queue library (VxFusion).....	181
<b>msgQDistShow</b>	– distributed message queue show routines (VxFusion) .....	182
<b>msgQEvLib</b>	– VxWorks events support for message queues.....	183
<b>msgQLib</b>	– message queue library .....	183
<b>msgQShow</b>	– message queue show routines.....	185
<b>msgQSmLib</b>	– shared memory message queue library (VxMP) .....	185
<b>muxLib</b>	– MUX network interface library .....	186
<b>muxTkLib</b>	– MUX toolkit Network Interface Library .....	188
<b>netBufLib</b>	– network buffer library .....	189
<b>netDrv</b>	– network remote file I/O driver .....	190
<b>netLib</b>	– network interface library.....	192
<b>netShow</b>	– network information display routines .....	192
<b>nfsdLib</b>	– Network File System (NFS) server library .....	193
<b>nfsDrv</b>	– Network File System (NFS) I/O driver .....	195
<b>nfsLib</b>	– Network File System (NFS) library .....	197
<b>ntPassFsLib</b>	– pass-through (to Windows NT) file system library .....	198
<b>passFsLib</b>	– pass-through (to UNIX) file system library (VxSim) .....	200
<b>pentiumALib</b>	– Pentium and PentiumPro specific routines.....	201
<b>pentiumLib</b>	– Pentium and Pentium[234] library .....	205
<b>pentiumShow</b>	– Pentium and Pentium[234] specific show routines.....	209
<b>pingLib</b>	– Packet InterNet Groper (PING) library.....	209
<b>pipeDrv</b>	– pipe I/O driver .....	210
<b>pppHookLib</b>	– PPP hook library.....	212
<b>pppLib</b>	– Point-to-Point Protocol library .....	212
<b>pppSecretLib</b>	– PPP authentication secrets library .....	214
<b>pppShow</b>	– Point-to-Point Protocol show routines.....	215
<b>proxyArpLib</b>	– proxy Address Resolution Protocol (ARP) server library .....	215
<b>proxyLib</b>	– proxy Address Resolution Protocol (ARP) client library .....	216
<b>pthreadLib</b>	– POSIX 1003.1c thread library interfaces.....	217
<b>ptyDrv</b>	– pseudo-terminal driver .....	223
<b>ramDiskCbio</b>	– RAM Disk Cached Block Driver .....	225
<b>ramDrv</b>	– RAM disk driver.....	225
<b>rawFsLib</b>	– raw block device file system library .....	226
<b>rBuffLib</b>	– dynamic ring buffer (rBuff) library .....	230
<b>rdiscLib</b>	– ICMP router discovery server library .....	230
<b>rebootLib</b>	– reboot support library .....	231
<b>remLib</b>	– remote command library.....	231
<b>remShellLib</b>	– remote access to target shell .....	232
<b>resolvLib</b>	– DNS resolver library .....	232
<b>ripLib</b>	– Routing Information Protocol (RIP) v1 and v2 library .....	234

<b>rlogLib</b>	– remote login library .....	236
<b>rngLib</b>	– ring buffer subroutine library .....	237
<b>routeEntryLib</b>	– route interface library for multiple matching entries .....	238
<b>routeLib</b>	– network route manipulation library .....	238
<b>routeMessageLib</b>	– message routines for the routing interface library .....	238
<b>rpcLib</b>	– Remote Procedure Call (RPC) support library .....	239
<b>rt11FsLib</b>	– RT-11 media-compatible file system library .....	239
<b>schedPxLib</b>	– scheduling library (POSIX) .....	245
<b>scsi1Lib</b>	– Small Computer System Interface (SCSI) library (SCSI-1) .....	246
<b>scsi2Lib</b>	– Small Computer System Interface (SCSI) library (SCSI-2) .....	249
<b>scsiCommonLib</b>	– SCSI library common commands for all devices (SCSI-2) .....	256
<b>scsiCtrlLib</b>	– SCSI thread-level controller library (SCSI-2) .....	256
<b>scsiDirectLib</b>	– SCSI library for direct access devices (SCSI-2) .....	257
<b>scsiLib</b>	– Small Computer System Interface (SCSI) library .....	258
<b>scsiMgrLib</b>	– SCSI manager library (SCSI-2) .....	259
<b>scsiSeqLib</b>	– SCSI sequential access device library (SCSI-2) .....	260
<b>selectLib</b>	– UNIX BSD 4.3 select library .....	261
<b>semBLib</b>	– binary semaphore library .....	262
<b>semCLib</b>	– counting semaphore library .....	264
<b>semEvLib</b>	– VxWorks events support for semaphores .....	265
<b>semLib</b>	– general semaphore library .....	266
<b>semMLib</b>	– mutual-exclusion semaphore library .....	268
<b>semOLib</b>	– release 4.x binary semaphore library .....	271
<b>semPxLib</b>	– semaphore synchronization library (POSIX) .....	271
<b>semPxShow</b>	– POSIX semaphore show library .....	273
<b>semShow</b>	– semaphore show routines .....	273
<b>semSmLib</b>	– shared memory semaphore library (VxMP) .....	274
<b>shellLib</b>	– shell execution routines .....	275
<b>sigLib</b>	– software signal facility library .....	276
<b>smMemLib</b>	– shared memory management library (VxMP) .....	281
<b>smMemShow</b>	– shared memory management show routines (VxMP) .....	284
<b>smNameLib</b>	– shared memory objects name database library (VxMP) .....	284
<b>smNameShow</b>	– shared memory objects name database show routines (VxMP) .....	286
<b>smNetLib</b>	– VxWorks interface to shared memory network (backplane) driver .....	287
<b>smNetShow</b>	– shared memory network driver show routines .....	288
<b>smObjLib</b>	– shared memory objects library (VxMP) .....	288
<b>smObjShow</b>	– shared memory objects show routines (VxMP) .....	291
<b>sntpcLib</b>	– Simple Network Time Protocol (SNTP) client library .....	291
<b>sntpsLib</b>	– Simple Network Time Protocol (SNTP) server library .....	292
<b>sockLib</b>	– generic socket library .....	293
<b>spyLib</b>	– spy CPU activity library .....	294
<b>symLib</b>	– symbol table subroutine library .....	295
<b>symSyncLib</b>	– host/target symbol table synchronization .....	297
<b>sysLib</b>	– system-dependent library .....	299
<b>tapeFsLib</b>	– tape sequential device file system library .....	302

<b>tarLib</b>	– UNIX tar compatible library .....	306
<b>taskArchLib</b>	– architecture-specific task management routines .....	307
<b>taskHookLib</b>	– task hook library .....	307
<b>taskHookShow</b>	– task hook show routines .....	308
<b>taskInfo</b>	– task information library .....	309
<b>taskLib</b>	– task management library .....	310
<b>taskShow</b>	– task show routines .....	312
<b>taskVarLib</b>	– task variables support library .....	313
<b>tcpShow</b>	– TCP information display routines .....	313
<b>telnetdLib</b>	– server library .....	314
<b>tffsConfig</b>	– TrueFFS configuration file for VxWorks .....	315
<b>tffsDrv</b>	– TrueFFS interface for VxWorks .....	316
<b>tftpdLib</b>	– Trivial File Transfer Protocol server library .....	318
<b>tftpLib</b>	– Trivial File Transfer Protocol (TFTP) client library .....	319
<b>tickLib</b>	– clock tick support library .....	321
<b>timerLib</b>	– timer library (POSIX) .....	322
<b>timexLib</b>	– execution timer facilities .....	323
<b>trgLib</b>	– trigger events control library .....	324
<b>trgShow</b>	– trigger show routine .....	325
<b>ttyDrv</b>	– provide terminal device access to serial channels .....	326
<b>tyLib</b>	– <i>tty</i> driver support library .....	326
<b>udpShow</b>	– UDP information display routines .....	332
<b>unixDrv</b>	– UNIX-file disk driver (VxSim for Solaris and VxSim for HP) .....	332
<b>unldLib</b>	– object module unloading library .....	334
<b>usrAta</b>	– ATA/ATAPI initialization .....	335
<b>usrConfig</b>	– user-defined system configuration library .....	336
<b>usrFd</b>	– floppy disk initialization .....	336
<b>usrFdiskPartLib</b>	– FDISK-style partition handler .....	337
<b>usrFsLib</b>	– file system user interface subroutine library .....	339
<b>usrIde</b>	– IDE initialization .....	340
<b>usrLib</b>	– user interface subroutine library .....	341
<b>usrScsi</b>	– SCSI initialization .....	342
<b>vmBaseLib</b>	– base virtual memory support library .....	343
<b>vmLib</b>	– architecture-independent virtual memory support library (VxVMI) .....	343
<b>vmShow</b>	– virtual memory show routines (VxVMI) .....	346
<b>vxLib</b>	– miscellaneous support routines .....	346
<b>wdbLib</b>	– WDB agent context management library .....	348
<b>wdbUserEvtLib</b>	– WDB user event library .....	348
<b>wdLib</b>	– watchdog timer library .....	349
<b>wdShow</b>	– watchdog show routines .....	350
<b>wvFileUploadPathLib</b>	– file destination for event data .....	350
<b>wvLib</b>	– event logging control library (WindView) .....	351
<b>wvNetLib</b>	– WindView for Networking Interface Library .....	356
<b>wvSockUploadPathLib</b>	– socket upload path library .....	357
<b>wvTmrLib</b>	– timer library (WindView) .....	357

<b>wvTsfsUploadPathLib</b>	– target host connection library using TSFS .....	358
<b>zbufLib</b>	– zbuf interface library .....	359
<b>zbufSockLib</b>	– zbuf socket interface library .....	361



---

## aioPxLib

<b>NAME</b>	<b>aioPxLib</b> – asynchronous I/O (AIO) library (POSIX)
<b>ROUTINES</b>	<b>aioPxLibInit()</b> - initialize the asynchronous I/O (AIO) library <b>aio_read()</b> - initiate an asynchronous read (POSIX) <b>aio_write()</b> - initiate an asynchronous write (POSIX) <b>lio_listio()</b> - initiate a list of asynchronous I/O requests (POSIX) <b>aio_suspend()</b> - wait for asynchronous I/O request(s) (POSIX) <b>aio_error()</b> - retrieve error status of asynchronous I/O operation (POSIX) <b>aio_return()</b> - retrieve return status of asynchronous I/O operation (POSIX)
<b>DESCRIPTION</b>	<p>This library implements asynchronous I/O (AIO) according to the definition given by the POSIX standard 1003.1b (formerly 1003.4, Draft 14). AIO provides the ability to overlap application processing and I/O operations initiated by the application. With AIO, a task can perform I/O simultaneously to a single file multiple times or to multiple files.</p> <p>After an AIO operation has been initiated, the AIO proceeds in logical parallel with the processing done by the application. The effect of issuing an asynchronous I/O request is as if a separate thread of execution were performing the requested I/O.</p>
<b>AIO LIBRARY</b>	The AIO library is initialized by calling <b>aioPxLibInit()</b> , which should be called once (typically at system start-up) after the I/O system has already been initialized.
<b>AIO COMMANDS</b>	<p>The file to be accessed asynchronously is opened via the standard open call. Open returns a file descriptor which is used in subsequent AIO calls.</p> <p>The caller initiates asynchronous I/O via one of the following routines:</p> <p><b>aio_read()</b> initiates an asynchronous read</p> <p><b>aio_write()</b> initiates an asynchronous write</p> <p><b>lio_listio()</b> initiates a list of asynchronous I/O requests</p> <p>Each of these routines has a return value and error value associated with it; however, these values indicate only whether the AIO request was successfully submitted (queued), not the ultimate success or failure of the AIO operation itself.</p> <p>There are separate return and error values associated with the success or failure of the AIO operation itself. The error status can be retrieved using <b>aio_error()</b>; however, until the AIO operation completes, the error status will be <b>EINPROGRESS</b>. After the AIO operation completes, the return status can be retrieved with <b>aio_return()</b>.</p>

The **aio\_cancel()** call cancels a previously submitted AIO request. The **aio\_suspend()** call waits for an AIO operation to complete.

Finally, the **aioShow()** call (not a standard POSIX function) displays outstanding AIO requests.

#### **AIO CONTROL BLOCK**

Each of the calls described above takes an AIO control block (**aiocb**) as an argument. The calling routine must allocate space for the **aiocb**, and this space must remain available for the duration of the AIO operation. (Thus the **aiocb** must not be created on the task's stack unless the calling routine will not return until after the AIO operation is complete and **aio\_return()** has been called.) Each **aiocb** describes a single AIO operation. Therefore, simultaneous asynchronous I/O operations using the same **aiocb** are not valid and produce undefined results.

The **aiocb** structure and the data buffers referenced by it are used by the system to perform the AIO request. Therefore, once the **aiocb** has been submitted to the system, the application must not modify the **aiocb** structure until after a subsequent call to **aio\_return()**. The **aio\_return()** call retrieves the previously submitted AIO data structures from the system. After the **aio\_return()** call, the calling application can modify the **aiocb**, free the memory it occupies, or reuse it for another AIO call.

As a result, if space for the **aiocb** is allocated off the stack the task should not be deleted (or complete running) until the **aiocb** has been retrieved from the system via an **aio\_return()**.

The **aiocb** is defined in **aio.h**. It has the following elements:

```
struct
{
    int          aio_fildes;
    off_t       aio_offset;
    volatile void * aio_buf;
    size_t      aio_nbytes;
    int         aio_reqprio;
    struct sigevent aio_sigevent;
    int         aio_lio_opcode;
    AIO_SYS     aio_sys;
} aiocb
```

#### **aio\_fildes**

file descriptor for I/O.

#### **aio\_offset**

offset from the beginning of the file where the AIO takes place. Note that performing AIO on the file does not cause the offset location to automatically increase as in read and write; the caller must therefore keep track of the location of reads and writes made to the file (see *POSIX COMPLIANCE* below).

**aio\_buf**

address of the buffer from/to which AIO is requested.

**aio\_nbytes**

number of bytes to read or write.

**aio\_reqprio**

amount by which to lower the priority of an AIO request. Each AIO request is assigned a priority; this priority, based on the calling task's priority, indicates the desired order of execution relative to other AIO requests for the file. The **aio\_reqprio** member allows the caller to lower (but not raise) the AIO operation priority by the specified value. Valid values for **aio\_reqprio** are in the range of zero through **AIO\_PRIO\_DELTA\_MAX**. If the value specified by **aio\_req\_prio** results in a priority lower than the lowest possible task priority, the lowest valid task priority is used.

**aio\_sigevent**

(optional) if nonzero, the signal to return on completion of an operation.

**aio\_lio\_opcode**

operation to be performed by a **lio\_listio()** call; valid entries include **LIO\_READ**, **LIO\_WRITE**, and **LIO\_NOP**.

**aio\_sys**

a Wind River Systems addition to the **aio\_cb** structure; it is used internally by the system and must not be modified by the user.

**EXAMPLES**

A writer could be implemented as follows:

```

if ((pAioWrite = calloc (1, sizeof (struct aio_cb))) == NULL)
    {
        printf ("calloc failed\n");
        return (ERROR);
    }
pAioWrite->aio_fildes = fd;
pAioWrite->aio_buf = buffer;
pAioWrite->aio_offset = 0;
strcpy (pAioWrite->aio_buf, "test string");
pAioWrite->aio_nbytes = strlen ("test string");
pAioWrite->aio_sigevent.sigev_notify = SIGEV_NONE;
aio_write (pAioWrite);
/* .
.
do other work
.
.
*/
/* now wait until I/O finishes */
while (aio_error (pAioWrite) == EINPROGRESS)
    taskDelay (1);

```

```
aio_return (pAioWrite);  
free (pAioWrite);
```

A reader could be implemented as follows:

```
/* initialize signal handler */  
  
action1.sa_sigaction = sigHandler;  
action1.sa_flags     = SA_SIGINFO;  
sigemptyset(&action1.sa_mask);  
sigaction (TEST_RT_SIG1, &action1, NULL);  
if ((pAioRead = calloc (1, sizeof (struct aiocb))) == NULL)  
{  
    printf ("calloc failed\n");  
    return (ERROR);  
}  
  
pAioRead->aio_fildes = fd;  
pAioRead->aio_buf     = buffer;  
pAioRead->aio_nbytes  = BUF_SIZE;  
pAioRead->aio_sigevent.sigev_signo = TEST_RT_SIG1;  
pAioRead->aio_sigevent.sigev_notify = SIGEV_SIGNAL;  
pAioRead->aio_sigevent.sigev_value.sival_ptr = (void *)pAioRead;  
  
aio_read (pAioRead);  
/*  
    .  
    .  
    do other work  
    .  
    .  
*/
```

The signal handler might look like the following:

```
void sigHandler  
(  
    int          sig,  
    struct siginfo info,  
    void *       pContext  
)  
{  
    struct aiocb * pAioDone;  
    pAioDone = (struct aiocb *) info.si_value.sival_ptr;  
    aio_return (pAioDone);  
    free (pAioDone);  
}
```

**POSIX COMPLIANCE**

Currently VxWorks does not support the **O\_APPEND** flag in the open call. Therefore, the user must keep track of the offset in the file that the asynchronous writes occur (as in the case of reads). The **aio\_offset** field is used to specify that file position.

In addition, VxWorks does not currently support synchronized I/O.

**INCLUDE FILES**     **aio.h**

**SEE ALSO**            POSIX 1003.1b document

---

## **aioPxShow**

**NAME**                **aioPxShow** – asynchronous I/O (AIO) show library

**ROUTINES**           **aioShow()** - show AIO requests

**DESCRIPTION**        This library implements the show routine for **aioPxLib**.

---

## **aioSysDrv**

**NAME**                **aioSysDrv** – AIO system driver

**ROUTINES**           **aioSysInit()** - initialize the AIO system driver

**DESCRIPTION**        This library is the AIO system driver. The system driver implements asynchronous I/O with system AIO tasks performing the AIO requests in a synchronous manner. It is installed as the default driver for AIO.

**SEE ALSO**            POSIX 1003.1b document

---

## **ansiAssert**

**NAME**                **ansiAssert** – ANSI **assert** documentation

**ROUTINES**           **assert()** - put diagnostics into programs (ANSI)

**DESCRIPTION** The header **assert.h** defines the **assert()** macro and refers to another macro, **NDEBUG**, which is not defined by **assert.h**. If **NDEBUG** is defined as a macro at the point in the source file where **assert.h** is included, the **assert()** macro is defined simply as:

```
#define assert(ignore) ((void)0)
```

ANSI specifies that **assert()** should be implemented as a macro, not as a routine. If the macro definition is suppressed in order to access an actual routine, the behavior is undefined.

**INCLUDE FILES** **stdio.h, stdlib.h, assert.h**

**SEE ALSO** American National Standard X3.159-1989

---

## ansiCtype

**NAME** **ansiCtype** – ANSI ctype documentation

**ROUTINES**

- isalnum()** - test whether a character is alphanumeric (ANSI)
- isalpha()** - test whether a character is a letter (ANSI)
- isctrl()** - test whether a character is a control character (ANSI)
- isdigit()** - test whether a character is a decimal digit (ANSI)
- isgraph()** - test whether a character is a printing, non-white-space character (ANSI)
- islower()** - test whether a character is a lower-case letter (ANSI)
- isprint()** - test whether a character is printable, including the space character (ANSI)
- ispunct()** - test whether a character is punctuation (ANSI)
- isspace()** - test whether a character is a white-space character (ANSI)
- isupper()** - test whether a character is an upper-case letter (ANSI)
- isxdigit()** - test whether a character is a hexadecimal digit (ANSI)
- tolower()** - convert an upper-case letter to its lower-case equivalent (ANSI)
- toupper()** - convert a lower-case letter to its upper-case equivalent (ANSI)

**DESCRIPTION** The header **ctype.h** declares several functions useful for testing and mapping characters. In all cases, the argument is an **int**, the value of which is representable as an **unsigned char** or is equal to the value of the macro **EOF**. If the argument has any other value, the behavior is undefined.

The behavior of the **ctype** functions is affected by the current locale. VxWorks supports only the "C" locale.

The term "printing character" refers to a member of an implementation-defined set of characters, each of which occupies one printing position on a display device; the term "control character" refers to a member of an implementation-defined set of characters that are not printing characters.

**INCLUDE FILES** `cctype.h`

**SEE ALSO** American National Standard X3.159-1989

---

## ansiLocale

**NAME** `ansiLocale` – ANSI `locale` documentation

**ROUTINES** `localeconv()` - set the components of an object with type `lconv` (ANSI)  
`setlocale()` - set the appropriate locale (ANSI)

**DESCRIPTION** The header `locale.h` declares two functions and one type, and defines several macros. The type is:

**struct lconv**  
contains members related to the formatting of numeric values. The structure should contain at least the members defined in `locale.h`, in any order.

**SEE ALSO** `localeconv()`, `setlocale()`, American National Standard X3.159-1989

---

## ansiMath

**NAME** `ansiMath` – ANSI `math` documentation

**ROUTINES** `asin()` - compute an arc sine (ANSI)  
`acos()` - compute an arc cosine (ANSI)  
`atan()` - compute an arc tangent (ANSI)  
`atan2()` - compute the arc tangent of  $y/x$  (ANSI)  
`ceil()` - compute the smallest integer greater than or equal to a specified value (ANSI)  
`cosh()` - compute a hyperbolic cosine (ANSI)  
`exp()` - compute an exponential value (ANSI)  
`fabs()` - compute an absolute value (ANSI)  
`floor()` - compute the largest integer less than or equal to a specified value (ANSI)  
`fmod()` - compute the remainder of  $x/y$  (ANSI)  
`frexp()` - break a floating-point number into a normalized fraction and power of 2 (ANSI)  
`ldexp()` - multiply a number by an integral power of 2 (ANSI)  
`log()` - compute a natural logarithm (ANSI)  
`log10()` - compute a base-10 logarithm (ANSI)

**ansiSetjmp**

**modf()** - separate a floating-point number into integer and fraction parts (ANSI)

**pow()** - compute the value of a number raised to a specified power (ANSI)

**sin()** - compute a sine (ANSI)

**cos()** - compute a cosine (ANSI)

**sinh()** - compute a hyperbolic sine (ANSI)

**sqrt()** - compute a non-negative square root (ANSI)

**tan()** - compute a tangent (ANSI)

**tanh()** - compute a hyperbolic tangent (ANSI)

**DESCRIPTION**

The header **math.h** declares several mathematical functions and defines one macro. The functions take double arguments and return double values.

The macro defined is:

**HUGE\_VAL**

expands to a positive double expression, not necessarily representable as a float.

The behavior of each of these functions is defined for all representable values of their input arguments. Each function executes as if it were a single operation, without generating any externally visible exceptions.

For all functions, a domain error occurs if an input argument is outside the domain over which the mathematical function is defined. The description of each function lists any applicable domain errors. On a domain error, the function returns an implementation-defined value; the value **EDOM** is stored in **errno**.

Similarly, a range error occurs if the result of the function cannot be represented as a double value. If the result overflows (the magnitude of the result is so large that it cannot be represented in an object of the specified type), the function returns the value **HUGE\_VAL**, with the same sign (except for the **tan()** function) as the correct value of the function; the value **ERANGE** is stored in **errno**. If the result underflows (the type), the function returns zero; whether the integer expression **errno** acquires the value **ERANGE** is implementation defined.

**INCLUDE FILES**

**math.h**

**SEE ALSO**

**mathALib**, American National Standard X3.159-1989

---

## ansiSetjmp

**NAME**

**ansiSetjmp** – ANSI **setjmp** documentation

**ROUTINES**

**setjmp()** - save the calling environment in a **jmp\_buf** argument (ANSI)

**longjmp()** - perform non-local goto by restoring saved environment (ANSI)

**DESCRIPTION** The header **setjmp.h** defines functions and one type for bypassing the normal function call and return discipline.

The type declared is:

**jmp\_buf**

an array type suitable for holding the information needed to restore a calling environment.

The ANSI C standard does not specify whether **setjmp()** is a subroutine or a macro.

**SEE ALSO** American National Standard X3.159-1989

---

## ansiStdarg

**NAME** **ansiStdarg** – ANSI **stdarg** documentation

**ROUTINES** **va\_start()** - initialize a **va\_list** object for use by **va\_arg()** and **va\_end()**  
**va\_arg()** - expand to an expression having the type and value of the call's next argument  
**va\_end()** - facilitate a normal return from a routine using a **va\_list** object

**DESCRIPTION** The header **stdarg.h** declares a type and defines three macros for advancing through a list of arguments whose number and types are not known to the called function when it is translated.

A function may be called with a variable number of arguments of varying types. The rightmost parameter plays a special role in the access mechanism, and is designated *parmN* in this description.

The type declared is:

**va\_list**

a type suitable for holding information needed by the macros **va\_start()**, **va\_arg()**, and **va\_end()**.

To access the varying arguments, the called function shall declare an object having type **va\_list**. The object (referred to here as *ap*) may be passed as an argument to another function; if that function invokes the **va\_arg()** macro with parameter *ap*, the value of *ap* in the calling function is indeterminate and is passed to the **va\_end()** macro prior to any further reference to *ap*.

**va\_start()** and **va\_arg()** have been implemented as macros, not as functions. The **va\_start()** and **va\_end()** macros should be invoked in the function accepting a varying number of arguments, if access to the varying arguments is desired.

The use of these macros is documented here as if they were architecture-generic. However, depending on the compilation environment, different macro versions are included by **vxWorks.h**.

**SEE ALSO**

American National Standard X3.159-1989

---

## ansiStdio

**NAME**

**ansiStdio** – ANSI **stdio** documentation

**ROUTINES**

**clearerr()** - clear end-of-file and error flags for a stream (ANSI)  
**fclose()** - close a stream (ANSI)  
**fdopen()** - open a file specified by a file descriptor (POSIX)  
**feof()** - test the end-of-file indicator for a stream (ANSI)  
**ferror()** - test the error indicator for a file pointer (ANSI)  
**fflush()** - flush a stream (ANSI)  
**fgetc()** - return the next character from a stream (ANSI)  
**fgetpos()** - store the current value of the file position indicator for a stream (ANSI)  
**fgets()** - read a specified number of characters from a stream (ANSI)  
**fileno()** - return the file descriptor for a stream (POSIX)  
 **fopen()** - open a file specified by name (ANSI)  
**fprintf()** - write a formatted string to a stream (ANSI)  
**fputc()** - write a character to a stream (ANSI)  
**fputs()** - write a string to a stream (ANSI)  
**fread()** - read data into an array (ANSI)  
**freopen()** - open a file specified by name (ANSI)  
**fscanf()** - read and convert characters from a stream (ANSI)  
**fseek()** - set the file position indicator for a stream (ANSI)  
**fsetpos()** - set the file position indicator for a stream (ANSI)  
**ftell()** - return the current value of the file position indicator for a stream (ANSI)  
**fwrite()** - write from a specified array (ANSI)  
**getc()** - return the next character from a stream (ANSI)  
**getchar()** - return the next character from the standard input stream (ANSI)  
**gets()** - read characters from the standard input stream (ANSI)  
**getw()** - read the next word (32-bit integer) from a stream  
**perror()** - map an error number in **errno** to an error message (ANSI)  
**putc()** - write a character to a stream (ANSI)  
**putchar()** - write a character to the standard output stream (ANSI)  
**puts()** - write a string to the standard output stream (ANSI)  
**putw()** - write a word (32-bit integer) to a stream  
**rewind()** - set the file position indicator to the beginning of a file (ANSI)  
**scanf()** - read and convert characters from the standard input stream (ANSI)

**setbuf()** - specify the buffering for a stream (ANSI)  
**setbuffer()** - specify buffering for a stream  
**setlinebuf()** - set line buffering for standard output or standard error  
**setvbuf()** - specify buffering for a stream (ANSI)  
**stdioInit()** - initialize standard I/O support  
**stdioFp()** - return the standard input/output/error FILE of the current task  
**stdioShowInit()** - initialize the standard I/O show facility  
**stdioShow()** - display file pointer internals  
**tmpfile()** - create a temporary binary file (Unimplemented) (ANSI)  
**tmpnam()** - generate a temporary file name (ANSI)  
**ungetc()** - push a character back into an input stream (ANSI)  
**fprintf()** - write a formatted string to a stream (ANSI)

**DESCRIPTION** The header **stdio.h** declares three types, several macros, and many functions for performing input and output.

#### Types

The types declared are **size\_t** and:

#### FILE

object type capable of recording all the information needed to control a stream, including its file position indicator, a pointer to its associated buffer (if any), an error indicator that records whether a read/write error has occurred, and an end-of-file indicator that records whether the end of the file has been reached.

#### fpos\_t

object type capable of recording all the information needed to specify uniquely every position within a file.

#### Macros

The macros are NULL and:

#### \_IOFBF, \_IOLBF, \_IONBF

expand to integral constant expressions with distinct values, suitable for use as the third argument to **setvbuf()**.

#### BUFSIZ

expands to an integral constant expression that is the size of the buffer used by **setbuf()**.

#### EOF

expands to a negative integral constant expression that is returned by several functions to indicate **end-of-file**, that is, no more input from a stream.

#### FOPEN\_MAX

expands to an integral constant expression that is the minimum number of the files that the system guarantees can be open simultaneously.

**FILENAME\_MAX**

expands to an integral constant expression that is the size needed for an array of **char** large enough to hold the longest file name string that can be used.

**L\_tmpnam**

expands to an integral constant expression that is the size needed for an array of **char** large enough to hold a temporary file name string generated by **tmpnam()**.

**SEEK\_CUR, SEEK\_END, SEEK\_SET**

expand to integral constant expressions with distinct values suitable for use as the third argument to **fseek()**.

**TMP\_MAX**

expands to an integral constant expression that is the minimum number of file names generated by **tmpnam()** that will be unique.

**'stderr, stdin, stdout'**

expressions of type "pointer to FILE" that point to the FILE objects associated, respectively, with the standard error, input, and output streams.

**STREAMS**

Input and output, whether to or from physical devices such as terminals and tape drives, or whether to or from files supported on structured storage devices, are mapped into logical data streams, whose properties are more uniform than their various inputs and outputs. Two forms of mapping are supported: for text streams and for binary streams.

A text stream is an ordered sequence of characters composed into lines, each line consisting of zero or more characters plus a terminating new-line character. Characters may have to be added, altered, or deleted on input and output to conform to differing conventions for representing text in the host environment. Thus, there is no need for a one-to-one correspondence between the characters in a stream and those in the external representation. Data read in from a text stream will necessarily compare equal to the data that were earlier written out to that stream only if: the data consists only of printable characters and the control characters horizontal tab and new-line; no new-line character is immediately preceded by space characters; and the last character is a new-line character. Space characters are written out immediately before a new-line character appears.

A binary stream is an ordered sequence of characters that can transparently record internal data. Data read in from a binary stream should compare equal to the data that was earlier written out to that stream, under the same implementation. However, such a stream may have a number of null characters appended to the end of the stream.

**Environmental Limits**

VxWorks supports text files with lines containing at least 254 characters, including the terminating new-line character. The value of the macro **BUFSIZ** is 1024.

**FILES**

A stream is associated with an external file (which may be a physical device) by opening a file, which may involve creating a new file. Creating an existing file causes its former contents to be discarded, if necessary. If a file can support positioning requests (such as a

disk file, as opposed to a terminal), then a file position indicator associated with the stream is positioned at the start (character number zero) of the file. The file position indicator is maintained by subsequent reads, writes, and positioning requests, to facilitate an orderly progression through the file. All input takes place as if characters were read by successive calls to **fgetc()**; all output takes place as if characters were written by successive calls to **fputc()**.

Binary files are not truncated, except as defined in **fopen()** documentation.

When a stream is unbuffered, characters are intended to appear from the source or at the destination as soon as possible. Otherwise characters may be accumulated and transmitted to or from the host environment as a block. When a stream is fully buffered, characters are intended to be transmitted to or from the host environment as a block when the buffer is filled. When a stream is line buffered, characters are intended to be transmitted to or from the host environment as a block when a new-line character is encountered. Furthermore, characters are intended to be transmitted as a block to the host environment when a buffer is filled, when input is requested on an unbuffered stream, or when input is requested on a line-buffered stream that requires the transmission of characters from the host environment. VxWorks supports these characteristics via the **setbuf()** and **setvbuf()** functions.

A file may be disassociated from a controlling stream by closing the file. Output streams are flushed (any unwritten buffer contents are transmitted to the host environment) before the stream is disassociated from the file. The value of a pointer to a **FILE** object is indeterminate after the associated file is closed (including the standard text streams).

The file may be subsequently reopened, by the same or another program execution, and its contents reclaimed or modified (if it can be repositioned at its start).

#### TASK TERMINATION

ANSI specifies that if the main function returns to its original caller or if **exit()** is called, all open files are closed (and hence all output streams are flushed) before program termination. This does **not** happen in VxWorks. The **exit()** function does not close all files opened for that task. A file opened by one task may be used and closed by another. Unlike in UNIX, when a VxWorks task exits, it is the responsibility of the task to **fclose()** its file pointers, except **stdin**, **stdout**, and **stderr**. If a task is to be terminated asynchronously, use **kill()** and arrange for a signal handler to clean up.

The address of the **FILE** object used to control a stream may be significant; a copy of a **FILE** object may not necessarily serve in place of the original.

At program startup, three text streams are predefined and need not be opened explicitly: standard input (for reading conventional input), standard output (for writing conventional output), and standard error (for writing diagnostic output). When opened, the standard error stream is not fully buffered; the standard input and standard output streams are fully buffered if and only if the stream can be determined not to refer to an interactive device.

Functions that open additional (non-temporary) files require a file name, which is a string. VxWorks allows the same file to be open multiple times simultaneously. It is up to the user to maintain synchronization between different tasks accessing the same file.

**FIOLIB** Several routines normally considered part of standard I/O -- **printf()**, **sprintf()**, **vprintf()**, **vsprintf()**, and **scanf()** -- are not implemented as part of the buffered standard I/O library; they are instead implemented in **fiolib**. They do not use the standard I/O buffering scheme. They are self-contained, formatted, but unbuffered I/O functions. This allows a limited amount of formatted I/O to be achieved without the overhead of the standard I/O library.

**SEE ALSO** **fiolib**, *American National Standard for Information Systems - Programming Language - C, ANSI X3.159-1989: Input/Output (**stdio.h**)*

---

## ansiStdlib

**NAME** **ansiStdlib** – ANSI **stdlib** documentation

**ROUTINES**

- abort()** - cause abnormal program termination (ANSI)
- abs()** - compute the absolute value of an integer (ANSI)
- atexit()** - call a function at program termination (Unimplemented) (ANSI)
- atof()** - convert a string to a **double** (ANSI)
- atoi()** - convert a string to an **int** (ANSI)
- atol()** - convert a string to a **long** (ANSI)
- bsearch()** - perform a binary search (ANSI)
- div()** - compute a quotient and remainder (ANSI)
- div\_r()** - compute a quotient and remainder (reentrant)
- labs()** - compute the absolute value of a **long** (ANSI)
- ldiv()** - compute the quotient and remainder of the division (ANSI)
- ldiv\_r()** - compute a quotient and remainder (reentrant)
- mblen()** - calculate the length of a multibyte character (Unimplemented) (ANSI)
- mbtowc()** - convert a multibyte character to a wide character (Unimplemented) (ANSI)
- wctomb()** - convert a wide character to a multibyte character (Unimplemented) (ANSI)
- mbstowcs()** - convert a series of multibyte char's to wide char's (Unimplemented) (ANSI)
- wcstombs()** - convert a series of wide char's to multibyte char's (Unimplemented) (ANSI)
- qsort()** - sort an array of objects (ANSI)
- rand()** - generate a pseudo-random integer between 0 and **RAND\_MAX** (ANSI)
- srand()** - reset the value of the seed used to generate random numbers (ANSI)
- strtod()** - convert the initial portion of a string to a double (ANSI)
- strtol()** - convert a string to a long integer (ANSI)

**strtoul()** - convert a string to an unsigned long integer (ANSI)  
**system()** - pass a string to a command processor (Unimplemented) (ANSI)

**DESCRIPTION** This library includes several standard ANSI routines. Note that where there is a pair of routines, such as **div()** and **div\_r()**, only the routine **xxx\_r()** is reentrant. The **xxx()** routine is not reentrant.

The header **stdlib.h** declares four types and several functions of general utility, and defines several macros.

#### Types

The types declared are **size\_t**, **wchar\_t**, and:

**div\_t**  
is the structure type of the value returned by the **div()**.

**ldiv\_t**  
is the structure type of the value returned by the **ldiv\_t()**.

#### Macros

The macros defined are **NULL** and:

**EXIT\_FAILURE**, **EXIT\_SUCCESS**  
expand to integral constant expressions that may be used as the argument to **exit()** to return unsuccessful or successful termination status, respectively, to the host environment.

**RAND\_MAX**  
expands to a positive integer expression whose value is the maximum number of bytes on a multibyte character for the extended character set specified by the current locale, and whose value is never greater than **MB\_LEN\_MAX**.

**INCLUDE FILES** **stdlib.h**

**SEE ALSO** American National Standard X3.159-1989

## ansiString

<b>NAME</b>	<b>ansiString</b> – ANSI <b>string</b> documentation
<b>ROUTINES</b>	<p><b>memchr()</b> - search a block of memory for a character (ANSI) <b>memcmp()</b> - compare two blocks of memory (ANSI) <b>memcpy()</b> - copy memory from one location to another (ANSI) <b>memmove()</b> - copy memory from one location to another (ANSI) <b>memset()</b> - set a block of memory (ANSI) <b>strcat()</b> - concatenate one string to another (ANSI) <b>strchr()</b> - find the first occurrence of a character in a string (ANSI) <b>strcmp()</b> - compare two strings lexicographically (ANSI) <b>strcoll()</b> - compare two strings as appropriate to <b>LC_COLLATE</b> (ANSI) <b>strcpy()</b> - copy one string to another (ANSI) <b>strcspn()</b> - return the string length up to the first character from a given set (ANSI) <b>strerror_r()</b> - map an error number to an error string (POSIX) <b>strerror()</b> - map an error number to an error string (ANSI) <b>strlen()</b> - determine the length of a string (ANSI) <b>strncat()</b> - concatenate characters from one string to another (ANSI) <b>strncmp()</b> - compare the first <i>n</i> characters of two strings (ANSI) <b>strncpy()</b> - copy characters from one string to another (ANSI) <b>strpbrk()</b> - find the first occurrence in a string of a character from a given set (ANSI) <b>strrchr()</b> - find the last occurrence of a character in a string (ANSI) <b>strspn()</b> - return the string length up to the first character not in a given set (ANSI) <b>strstr()</b> - find the first occurrence of a substring in a string (ANSI) <b>strtok()</b> - break down a string into tokens (ANSI) <b>strtok_r()</b> - break down a string into tokens (reentrant) (POSIX) <b>strxfrm()</b> - transform up to <i>n</i> characters of <i>s2</i> into <i>s1</i> (ANSI)</p>
<b>DESCRIPTION</b>	<p>This library includes several standard ANSI routines. Note that where there is a pair of routines, such as <b>did()</b> and <b>div_r()</b>, only the routine <b>xxx_r()</b> is reentrant. The <b>xxx()</b> routine is not reentrant.</p> <p>The header <b>string.h</b> declares one type and several functions, and defines one macro useful for manipulating arrays of character type and other objects treated as array of character type. The type is <b>size_t</b> and the macro <b>NULL</b>. Various methods are used for determining the lengths of the arrays, but in all cases a <b>char *</b> or <b>void *</b> argument points to the initial (lowest addressed) character of the array. If an array is accessed beyond the end of an object, the behavior is undefined.</p>
<b>SEE ALSO</b>	American National Standard X3.159-1989

---

# ansiTime

<b>NAME</b>	<b>ansiTime</b> – ANSI time documentation
<b>ROUTINES</b>	<b>asctime()</b> - convert broken-down time into a string (ANSI) <b>asctime_r()</b> - convert broken-down time into a string (POSIX) <b>clock()</b> - determine the processor time in use (ANSI) <b>ctime()</b> - convert time in seconds into a string (ANSI) <b>ctime_r()</b> - convert time in seconds into a string (POSIX) <b>difftime()</b> - compute the difference between two calendar times (ANSI) <b>gmtime()</b> - convert calendar time into UTC broken-down time (ANSI) <b>gmtime_r()</b> - convert calendar time into broken-down time (POSIX) <b>localtime()</b> - convert calendar time into broken-down time (ANSI) <b>localtime_r()</b> - convert calendar time into broken-down time (POSIX) <b>mktime()</b> - convert broken-down time into calendar time (ANSI) <b>strftime()</b> - convert broken-down time into a formatted string (ANSI) <b>time()</b> - determine the current calendar time (ANSI)
<b>DESCRIPTION</b>	The header <b>time.h</b> defines two macros and declares four types and several functions for manipulating time. Many functions deal with a <b>calendar time</b> that represents the current date (according to the Gregorian calendar) and time. Some functions deal with <b>local time</b> , which is the calendar time expressed for some specific time zone, and with Daylight Saving Time, which is a temporary change in the algorithm for determining local time. The local time zone and Daylight Saving Time are implementation-defined.
<b>Macros</b>	The macros defined are NULL and: <b>CLOCKS_PER_SEC</b> the number of ticks per second.
<b>Types</b>	The types declared are <b>size_t</b> and: <b>clock_t, time_t</b> arithmetic types capable of representing times. <b>struct tm</b> holds the components of a calendar time in what is known as “broken-down time.” The structure contains at least the following members, in any order. The semantics of the members and their normal ranges are expressed in the comments.

int **tm\_sec**; seconds after the minute - [0, 59]  
int **tm\_min**; minutes after the hour - [0, 59]  
int **tm\_hour**; hours after midnight - [0, 23]  
int **tm\_mday**; day of the month - [1, 31]  
int **tm\_mon**; months since January - [0, 11]  
int **tm\_year**; years since 1900  
int **tm\_wday**; days since Sunday - [0, 6]  
int **tm\_yday**; days since January 1 - [0, 365]  
int **tm\_isdst**; Daylight Saving Time flag

The value of **tm\_isdst** is positive if Daylight Saving Time is in effect, zero if Daylight Saving Time is not in effect, and negative if the information is not available.

If the environment variable TIMEZONE is set, the information is retrieved from this variable, otherwise from the locale information. TIMEZONE is of the form:

*name\_of\_zone*:<(unused)<:*time\_in\_minutes\_from\_UTC*:*daylight\_start*:*daylight\_end*

To calculate local time, the value of *time\_in\_minutes\_from\_UTC* is subtracted from UTC; *time\_in\_minutes\_from\_UTC* must be positive.

Daylight information is expressed as mmddhh (month-day-hour), for example:

**UTC::0:040102:100102**

**REENTRANCY** Where there is a pair of routines, such as **div()** and **div\_r()**, only the routine **xxx\_r()** is reentrant. The **xxx()** routine is not reentrant.

**INCLUDE FILES** **time.h**

**SEE ALSO** **ansiLocale**, American National Standard X3.159-1989

---

## arpLib

**NAME** **arpLib** – Address Resolution Protocol (ARP) table manipulation library

**ROUTINES** **arpAdd()** - create or modify an ARP table entry  
**arpDelete()** - remove an ARP table entry  
**arpFlush()** - flush all entries in the system ARP table  
**arpResolve()** - resolve a hardware address for a specified Internet address

**DESCRIPTION** This library provides direct access to the address translation table maintained by the Address Resolution Protocol (ARP). Each entry in the table maps an Internet Protocol (IP)

address to a physical hardware address. This library supports only those entries that translate between IP and Ethernet addresses. It is linked into the VxWorks image if **INCLUDE\_ARP** is defined at the time the image is built. The underlying ARP protocol, which creates and maintains the table, is included automatically as part of the IP component.

**RELATED INTERFACES**

The **arpShow()** routine (in the **netShow** library) displays the current contents of the ARP table.

A low -level interface to the ARP table is available with the socket-specific **SIOCSARP**, **SIOCDDARP** and **SIOCGARP** ioctl functions.

**INCLUDE FILES**     **arpLib.h**

**SEE ALSO**         **inetLib, routeLib, netShow**

---

## bLib

<b>NAME</b>	<b>bLib</b> – buffer manipulation library
<b>ROUTINES</b>	<b>bcmp()</b> - compare one buffer to another <b>binvert()</b> - invert the order of bytes in a buffer <b>bswap()</b> - swap buffers <b>swab()</b> - swap bytes <b>uswab()</b> - swap bytes with buffers that are not necessarily aligned <b>bzero()</b> - zero out a buffer <b>bcopy()</b> - copy one buffer to another <b>bcopyBytes()</b> - copy one buffer to another one byte at a time <b>bcopyWords()</b> - copy one buffer to another one word at a time <b>bcopyLongs()</b> - copy one buffer to another one long word at a time <b>bfill()</b> - fill a buffer with a specified character <b>bfillBytes()</b> - fill buffer with a specified character one byte at a time <b>index()</b> - find the first occurrence of a character in a string <b>rindex()</b> - find the last occurrence of a character in a string
<b>DESCRIPTION</b>	<p>This library contains routines to manipulate buffers of variable-length byte arrays. Operations are performed on long words when possible, even though the buffer lengths are specified in bytes. This occurs only when source and destination buffers start on addresses that are both odd or both even. If one buffer is even and the other is odd, operations must be done one byte at a time (because of alignment problems inherent in the MC68000), thereby slowing down the process.</p> <p>Certain applications, such as byte-wide memory-mapped peripherals, may require that only byte operations be performed. For this purpose, the routines <b>bcopyBytes()</b> and <b>bfillBytes()</b> provide the same functions as <b>bcopy()</b> and <b>bfill()</b>, but use only byte-at-a-time operations. These routines do not check for null termination.</p>
<b>INCLUDE FILES</b>	<b>string.h</b>
<b>SEE ALSO</b>	<b>ansiString</b>

---

## bootConfig

**B**

**NAME** **bootConfig** – system configuration module for boot ROMs

**ROUTINES** No Callable Routines

**DESCRIPTION** This is the WRS-supplied configuration module for the VxWorks boot ROM. It is a stripped-down version of **usrConfig.c**, having no VxWorks shell or debugging facilities. Its primary function is to load an object module over the network with either RSH or FTP. Additionally, a simple set of single letter commands is provided for displaying and modifying memory contents. Use this module as a starting point for placing applications in ROM.

---

## bootInit

**NAME** **bootInit** – ROM initialization module

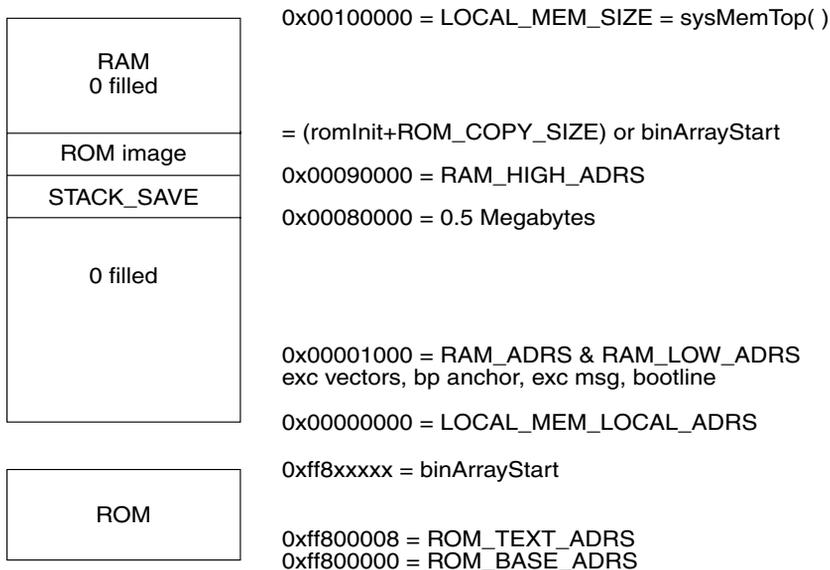
**ROUTINES** **romStart()** - generic ROM initialization

**DESCRIPTION** This module provides a generic boot ROM facility. The target-specific **romInit.s** module performs the minimal preliminary board initialization and then jumps to the C routine **romStart()**. This routine, still executing out of ROM, copies the first stage of the startup code to a RAM address and jumps to it. The next stage clears memory and then uncompresses the remainder of ROM into the final VxWorks ROM image in RAM.

A modified version of the Public Domain **zlib** library is used to uncompress the VxWorks boot ROM executable linked with it. Compressing object code typically achieves over 55% compression, permitting much larger systems to be burned into ROM. The only expense is the added few seconds delay while the first two stages complete.

### ROM AND RAM MEMORY LAYOUT

Example memory layout for a 1-megabyte board:



**SEE ALSO** **inflate()**, **romInit()**, and **deflate()**

**AUTHOR** The original compression software for **zlib** was written by Jean-Loup Gailly and Mark Adler. See the manual pages of **inflate** and **deflate** for more information on their freely available compression software.

---

## bootLib

<b>NAME</b>	<b>bootLib</b> – boot ROM subroutine library
<b>ROUTINES</b>	<p><b>bootStringToStruct()</b> - interpret the boot parameters from the boot line</p> <p><b>bootStructToString()</b> - construct a boot line</p> <p><b>bootParamsShow()</b> - display boot line parameters</p> <p><b>bootParamsPrompt()</b> - prompt for boot line parameters</p> <p><b>bootLeaseExtract()</b> - extract the lease information from an Internet address</p> <p><b>bootNetmaskExtract()</b> - extract the net mask field from an Internet address</p> <p><b>bootBpAnchorExtract()</b> - extract a backplane address from a device field</p>
<b>DESCRIPTION</b>	<p>This library contains routines for manipulating a boot line. Routines are provided to interpret, construct, print, and prompt for a boot line.</p> <p>When VxWorks is first booted, certain parameters can be specified, such as network addresses, boot device, host, and start-up file. This information is encoded into a single ASCII string known as the boot line. The boot line is placed at a known address (specified in <b>config.h</b>) by the boot ROMs so that the system being booted can discover the parameters that were used to boot the system. The boot line is the only means of communication from the boot ROMs to the booted system.</p> <p>The boot line is of the form:</p> <pre><b>bootdev</b> (<i>unitnum</i>, <i>procnum</i>)<i>hostname</i>:<i>filename</i> <b>e</b>=# <b>b</b>=# <b>h</b>=# <b>g</b>=# <b>u</b>=<i>userid</i> <b>pw</b>=<i>passwd</i> <b>f</b>=# <b>tn</b>=<i>targetname</i> <b>s</b>=<i>startupscript</i> <b>o</b>=<i>other</i></pre> <p>where:</p> <p><i>bootdev</i> the boot device (required); for example, "ex" for Excelan Ethernet, "bp" for backplane. For the backplane, this field can have an optional anchor address specification of the form "bp=<i>adrs</i>" (see <b>bootBpAnchorExtract()</b>).</p> <p><i>unitnum</i> the unit number of the boot device (0..n).</p> <p><i>procnum</i> the processor number on the backplane, 0..n (required for VME boards).</p> <p><i>hostname</i> the name of the boot host (required).</p> <p><i>filename</i> the file to be booted (required).</p> <p><b>e</b> the Internet address of the Ethernet interface. This field can have an optional subnet mask of the form <i>inet_adrs:subnet_mask</i>. If DHCP is used to obtain the configuration</p>

**bootLib**

parameters, lease timing information may also be present. This information takes the form *lease\_duration:lease\_origin* and is appended to the end of the field. (see **bootNetmaskExtract()** and **bootLeaseExtract()**).

**b**  
the Internet address of the backplane interface. This field can have an optional subnet mask and/or lease timing information as “e”.

**h**  
the Internet address of the boot host.

**g**  
the Internet address of the gateway to the boot host. Leave this parameter blank if the host is on same network.

**u**  
a valid user name on the boot host.

**pw**  
the password for the user on the host. This parameter is usually left blank. If specified, FTP is used for file transfers.

**f**  
the system-dependent configuration flags. This parameter contains an **or** of option bits defined in **sysLib.h**.

**tn**  
the name of the system being booted

**s**  
the name of a file to be executed as a start-up script.

**o**  
“other” string for use by the application.

The Internet addresses are specified in “dot” notation (*e.g.*, 90.0.0.2). The order of assigned values is arbitrary.

**EXAMPLE**      `enp(0,0)host:/usr/wpwr/target/config/mz7122/vxWorks e=90.0.0.2 b=91.0.0.2  
h=100.0.0.4 g=90.0.0.3 u=bob pw=realtime f=2 tn=target  
s=host:/usr/bob/startup o=any_string`

**INCLUDE FILES**      **bootLib.h**

**SEE ALSO**            **bootConfig**

---

# bootpLib

<b>NAME</b>	<b>bootpLib</b> – Bootstrap Protocol (BOOTP) client library
<b>ROUTINES</b>	<b>bootpLibInit()</b> - BOOTP client library initialization <b>bootpParamsGet()</b> - retrieve boot parameters using BOOTP <b>bootpMsgGet()</b> - send a BOOTP request message and retrieve reply
<b>DESCRIPTION</b>	This library implements the client side of the Bootstrap Protocol (BOOTP). This protocol allows a host to initialize automatically by obtaining its IP address, boot file name, and boot host's IP address over a network. The <b>bootpLibInit()</b> routine links this library into the VxWorks image. This happens automatically if <b>INCLUDE_BOOTP</b> is defined at the time the image is built.
<b>CONFIGURATION INTERFACE</b>	When used during boot time, the BOOTP library attempts to retrieve the required configuration information from a BOOTP server using the interface described below. If it is successful, the remainder of the boot process continues as if the information were entered manually.
<b>HIGH-LEVEL INTERFACE</b>	The <b>bootpParamsGet()</b> routine retrieves a set of configuration parameters according to the client-server interaction described in RFC 951 and clarified in RFC 1542. The parameter descriptor structure it accepts as an argument allows the retrieval of any combination of the options described in RFC 1533 (if supported by the BOOTP server and specified in the database). During the default system boot process, the routine obtains the boot file, the Internet address, and the host Internet address. It also obtains the subnet mask and the Internet address of an IP router, if available.
<b>LOW-LEVEL INTERFACE</b>	The <b>bootpMsgGet()</b> routine transmits an arbitrary BOOTP request message and provides direct access to any reply. This interface provides a method for supporting alternate BOOTP implementations which may not fully comply with the recommended behavior in RFC 1542. For example, it allows transmission of BOOTP messages to an arbitrary UDP port and provides access to the vendor-specific field to handle custom formats which differ from the RFC 1533 implementation. The <b>bootpParamsGet()</b> routine already extracts all options which that document defines.
<b>EXAMPLE</b>	The following code fragment demonstrates use of the BOOTP library:

```
#include "bootpLib.h"
#define _MAX_BOOTP_RETRIES 1
struct bootpParams bootParams;
struct in_addr      clntAddr;
```

```
struct in_addr      hostAddr;
char                bootFile [SIZE_FILE];
int                subnetMask;
struct in_addr_list routerList;
struct in_addr      gateway;
struct ifnet *      pIf;
/* Retrieve the interface descriptor of the transmitting device. */
pIf = ifunit ("ln0");
if (pIf == NULL)
{
    printf ("Device not found.\n");
    return (ERROR);
}
/* Setup buffers for information from BOOTP server. */
bzero ( (char *)&clntAddr, sizeof (struct in_addr));
bzero ( (char *)&hostAddr, sizeof (struct in_addr));
bzero (bootFile, SIZE_FILE);
subnetMask = 0;
bzero ( (char *)&gateway, sizeof (struct in_addr));
/* Set all pointers in parameter descriptor to NULL. */
bzero ((char *)&bootParams, sizeof (struct bootpParams));
/* Set pointers corresponding to desired options. */
bootParams.netmask = (struct in_addr *)&subnetMask;
routerlist.addr = &gateway;
routerlist.num = 1;
bootParams.routers = &routerlist;
/*
    @ Send request and wait for reply, retransmitting as necessary up to
    @ given limit. Copy supplied entries into buffers if reply received.
    */
result = bootpParamsGet (pIf, _MAX_BOOTP_RETRIES,
                        &clntAddr, &hostAddr, NULL, bootFile, &bootParams);
if (result != OK)
    return (ERROR);
```

**INCLUDE FILES**    bootpLib.h

**SEE ALSO**        RFC 951, RFC 1542, RFC 1533,

---

# bpfDrv

**NAME** **bpfDrv** – Berkeley Packet Filter (BPF) I/O driver library

**ROUTINES** **bpfDrv()** - initialize the BPF driver  
**bpfDevCreate()** - create Berkeley Packet Filter device  
**bpfDevDelete()** - destroy Berkeley Packet Filter device

**DESCRIPTION** This library provides a driver which supports the customized retrieval of incoming network data that meets the criteria imposed by a user-specified filter.

## USER-CALLABLE ROUTINES

The **bpfDrv()** routine initializes the driver and the **bpfDevCreate()** routine creates a packet filter device. Each BPF device allows direct access to the incoming data from one or more network interfaces.

## CREATING BPF DEVICES

In order to retrieve incoming network data, a BPF device must be created by calling the **bpfDevCreate()** routine:

```
STATUS bpfDevCreate
(  
    char * pDevName,      /* I/O system device name */  
    int numUnits,        /* number of device units */  
    int bufSize          /* block size for the BPF device */  
)
```

The *numUnits* parameter specifies the maximum number of BPF units for the device. Each unit is accessed through a separate file descriptor for use with a unique filter and/or a different network interface. For example, the following call creates the **/bpf0** and **/bpf1** units:

```
bpfDevCreate ("/bpf", 2, 4096);
```

## CONFIGURING BPF DEVICES

After opening a device unit, the associated file descriptor must be bound to a specific network interface with the **BIOCSETIF ioctl()** option. The **BIOCSETF ioctl()** option adds any filter instructions. Each file descriptor receives a copy of any data which matches the filter. Different file descriptors may share the same interface. The underlying filters will receive an identical data stream.

**IOCTL FUNCTIONS** The BPF driver supports the following **ioctl()** functions:

---

**NOTE:** When reading data from BPF units, the supplied buffer must be able to accept an entire block of data as defined by the *bufSize* parameter to the **bpfDevCreate()** routine. That value is also available with the **BIOCGLEN ioctl()** option described above.

---

**INCLUDE FILES**      **bpfDrv.h**

**SEE ALSO**            **ioLib**

---

## cache4kcLib

<b>NAME</b>	<b>cache4kcLib</b> – MIPS 4kc cache management library
<b>ROUTINES</b>	<b>cache4kcLibInit()</b> - initialize the 4kc cache library
<b>DESCRIPTION</b>	<p>This library contains architecture-specific cache library functions for the MIPS 4kc architecture. The 4kc utilizes a variable-size instruction and data cache that operates in write-through mode. Cache line size also varies.</p> <p>For general information about caching, see the manual entry for <b>cacheLib</b>.</p>
<b>INCLUDE FILES</b>	<b>cacheLib.h</b>
<b>SEE ALSO</b>	<b>cacheLib</b>

---

## cacheArchLib

<b>NAME</b>	<b>cacheArchLib</b> – architecture-specific cache management library
<b>ROUTINES</b>	<b>cacheArchLibInit()</b> - initialize the cache library <b>cacheArchClearEntry()</b> - clear an entry from a cache (68K, x86) <b>cacheStoreBufEnable()</b> - enable the store buffer (MC68060 only) <b>cacheStoreBufDisable()</b> - disable the store buffer (MC68060 only)
<b>DESCRIPTION</b>	<p>This library contains architecture-specific cache library functions for the following processor cache families: Motorola 68K, Intel 960, Intel x86, PowerPC, ARM, and the Solaris and Windows simulators. Each routine description indicates which architecture families support it. Within families, different members support different cache mechanisms; thus, some operations cannot be performed by certain processors because they lack particular functionalities. In such cases, the routines in this library return <b>ERROR</b>. Processor-specific constraints are addressed in the manual entries for routines in this library. If the caches are unavailable or uncontrollable, the routines return <b>ERROR</b>. The exception to this rule is the 68020; although the 68020 has no cache, data cache operations return <b>OK</b>.</p> <p>The MIPS architecture families have cache-related routines in individual BSP libraries. See the reference pages for the individual libraries and routines.</p>
<b>INCLUDE FILES</b>	<b>cacheLib.h</b> , <b>mmuLib.h</b> (ARM only)
<b>SEE ALSO</b>	<b>cacheLib</b> , <b>vmLib</b>

## cacheAuLib

<b>NAME</b>	<b>cacheAuLib</b> – Alchemy Au cache management library
<b>ROUTINES</b>	<b>cacheAuLibInit()</b> - initialize the Au cache library
<b>DESCRIPTION</b>	This library contains architecture-specific cache library functions for the Alchemy Au architecture. The Au utilizes a variable-size instruction and data cache that operates in write-through mode. Cache line size also varies.  For general information about caching, see the manual entry for <b>cacheLib</b> .
<b>INCLUDE FILES</b>	<b>cacheLib.h</b>
<b>SEE ALSO</b>	<b>cacheLib</b>

---

## cacheLib

<b>NAME</b>	<b>cacheLib</b> – cache management library
<b>ROUTINES</b>	<b>cacheLibInit()</b> - initialize the cache library for a processor architecture <b>cacheEnable()</b> - enable the specified cache <b>cacheDisable()</b> - disable the specified cache <b>cacheLock()</b> - lock all or part of a specified cache <b>cacheUnlock()</b> - unlock all or part of a specified cache <b>cacheFlush()</b> - flush all or some of a specified cache <b>cacheInvalidate()</b> - invalidate all or some of a specified cache <b>cacheClear()</b> - clear all or some entries from a cache <b>cachePipeFlush()</b> - flush processor write buffers to memory <b>cacheTextUpdate()</b> - synchronize the instruction and data caches <b>cacheDmaMalloc()</b> - allocate a cache-safe buffer for DMA devices and drivers <b>cacheDmaFree()</b> - free the buffer acquired with <b>cacheDmaMalloc()</b> <b>cacheDrvFlush()</b> - flush the data cache for drivers <b>cacheDrvInvalidate()</b> - invalidate data cache for drivers <b>cacheDrvVirtToPhys()</b> - translate a virtual address for drivers <b>cacheDrvPhysToVirt()</b> - translate a physical address for drivers
<b>DESCRIPTION</b>	This library provides architecture-independent routines for managing the instruction and data caches. Architecture-dependent routines are documented in the architecture-specific libraries.

The cache library is initialized by `cacheLibInit()` in `usrInit()`. The `cacheLibInit()` routine typically calls an architecture-specific initialization routine in one of the architecture-specific libraries. The initialization routine places the cache in a known and quiescent state, ready for use, but not yet enabled. Cache devices are enabled and disabled by calls to `cacheEnable()` and `cacheDisable()`, respectively.

The structure `CACHE_LIB` in `cacheLib.h` provides a function pointer that allows for the installation of different cache implementations in an architecture-independent manner. If the processor family allows more than one cache implementation, the board support package (BSP) must select the appropriate cache library using the function pointer `sysCacheLibInit`. The `cacheLibInit()` routine calls the initialization function attached to `sysCacheLibInit` to perform the actual `CACHE_LIB` function pointer initialization (see `cacheLib.h`). Note that `sysCacheLibInit` must be initialized when declared; it need not exist for architectures with a single cache design. Systems without caches have all `NULL` pointers in the `CACHE_LIB` structure. For systems with bus snooping, NULLifying the flush and invalidate function pointers in `sysHwInit()` improves overall system and driver performance.

Function pointers also provide a way to supplement the cache library or attach user-defined cache functions for managing secondary cache systems.

Parameters specified by `cacheLibInit()` are used to select the cache mode, either write-through (`CACHE_WRITETHROUGH`) or copyback (`CACHE_COPYBACK`), as well as to implement all other cache configuration features via software bit-flags. Note that combinations, such as setting copyback and write-through at the same time, do not make sense.

Typically, the first argument passed to cache routines after initialization is the `CACHE_TYPE`, which selects the data cache (`DATA_CACHE`) or the instruction cache (`INSTRUCTION_CACHE`).

Several routines accept two additional arguments: an address and the number of bytes. Some cache operations can be applied to the entire cache (bytes = `ENTIRE_CACHE`) or to a portion of the cache. This range specification allows the cache to be selectively locked, unlocked, flushed, invalidated, and cleared. The two complementary routines, `cacheDmaMalloc()` and `cacheDmaFree()`, are tailored for efficient driver writing. The `cacheDmaMalloc()` routine attempts to return a “cache-safe” buffer, which is created by the MMU and a set of flush and invalidate function pointers. Examples are provided below in the section “Using the Cache Library.”

Most routines in this library return a `STATUS` value of `OK`, or `ERROR` if the cache selection is invalid or the cache operation fails.

## BACKGROUND

The emergence of RISC processors and effective CISC caches has made cache and MMU support a key enhancement to VxWorks. (For more information about MMU support, see the manual entry for `vmLib`.) The VxWorks cache strategy is to maintain coherency between the data cache and RAM and between the instruction and data caches. VxWorks also preserves overall system performance. The product is designed to support several



The instruction cache and data cache (2) can get out of sync when the loader, the debugger, and the interrupt connection routines are being used. The instructions resulting from these operations are loaded into the data cache, but not necessarily the instruction cache, in which case there is a coherency problem. This can be fixed by “flushing” the data cache entries to RAM, then “invalidating” the instruction cache entries. The invalid instruction cache tags will force the retrieval of the new instructions that the data cache has just flushed to RAM.

Cache lines that are shared (3) by more than one task create coherency problems. These are manifest when one thread of execution invalidates a cache line in which entries may belong to another thread. This can be avoided by allocating memory on a cache line boundary, then rounding up to a multiple of the cache line size.

The best way to preserve cache coherency with optimal performance (Harvard architecture, copyback mode, no software intervention) is to use hardware with bus snooping capabilities. The caches, the RAM, the DMA devices, and all other bus masters are tied to a physical bus where the caches can “snoop” or watch the bus transactions. The address cycle and control (read/write) bits are broadcast on the bus to allow snooping. Data transfer cycles are deferred until absolutely necessary. When one of the entries on the physical side of the cache is modified by an asynchronous action, the cache(s) marks its entry(s) as invalid. If an access is made by the processor (logical side) to the now invalid cached entry, it is forced to retrieve the valid entry from RAM. If while in copyback mode the processor writes to a cached entry, the RAM version becomes stale. If another master attempts to access that stale entry in RAM, the cache with the valid version preempts the access and writes the valid data to RAM. The interrupted access then restarts and retrieves the now-valid data in RAM. Note that this configuration allows only one valid entry at any time. At this time, only a few boards provide the snooping capability; therefore, cache support software must be designed to handle incoherency hazards without degrading performance.

The determinism, interrupt latency, and benchmarks for a cached system are exceedingly difficult to specify (best case, worst case, average case) due to cache hits and misses, line flushes and fills, atomic burst cycles, global and local instruction and data cache locking, copyback versus write-through modes, hardware coherency support (or lack of), and MMU operations (table walks, TLB locking).

#### USING THE CACHE LIBRARY

The coherency problems described above can be overcome by adding cache support to existing software. For code segments that are not time-critical (loader, debugger, interrupt connection), the following sequence should be used first to flush the data cache entries and then to invalidate the corresponding instruction cache entries.

```
cacheFlush (DATA_CACHE, address, bytes);  
cacheInvalidate (INSTRUCTION_CACHE, address, bytes);
```

For time-critical code, implementation is up to the driver writer. The following are tips for using the VxWorks cache library effectively.

Incorporate cache calls in the driver program to maintain overall system performance. The cache may be disabled to facilitate driver development; however, high-performance production systems should operate with the cache enabled. A disabled cache will dramatically reduce system performance for a completed application.

Buffers can be static or dynamic. Mark buffers “non-cacheable” to avoid cache coherency problems. This usually requires MMU support. Dynamic buffers are typically smaller than their static counterparts, and they are allocated and freed often. When allocating either type of buffer, it should be designated non-cacheable; however, dynamic buffers should be marked “cacheable” before being freed. Otherwise, memory becomes fragmented with numerous non-cacheable dynamic buffers.

Alternatively, use the following flush/invalidate scheme to maintain cache coherency.

```
cacheInvalidate (DATA_CACHE, address, bytes); /* input buffer */
cacheFlush (DATA_CACHE, address, bytes); /* output buffer */
```

The principle is to flush output buffers before each use and invalidate input buffers before each use. Flushing only writes modified entries back to RAM, and instruction cache entries never get modified.

Several flush and invalidate macros are defined in **cacheLib.h**. Since optimized code uses these macros, they provide a mechanism to avoid unnecessary cache calls and accomplish the necessary work (return OK). Needless work includes flushing a write-through cache, flushing or invalidating cache entries in a system with bus snooping, and flushing or invalidating cache entries in a system without caches. The macros are set to reflect the state of the cache system hardware and software. Example 1 The following example is of a simple driver that uses **cacheFlush()** and **cacheInvalidate()** from the cache library to maintain coherency and performance. There are two buffers (lines 3 and 4), one for input and one for output. The output buffer is obtained by the call to **memalign()**, a special version of the well-known **malloc()** routine (line 6). It returns a pointer that is rounded down and up to the alignment parameter’s specification. Note that cache lines should not be shared, therefore **\_CACHE\_ALIGN\_SIZE** is used to force alignment. If the memory allocator fails (line 8), the driver will typically return **ERROR** (line 9) and quit.

The driver fills the output buffer with initialization information, device commands, and data (line 11), and is prepared to pass the buffer to the device. Before doing so the driver must flush the data cache (line 13) to ensure that the buffer is in memory, not hidden in the cache. The **drvWrite()** routine lets the device know that the data is ready and where in memory it is located (line 14).

More driver code is executed (line 16), then the driver is ready to receive data that the device has placed in an input buffer in memory (line 18). Before the driver can work with the incoming data, it must invalidate the data cache entries (line 19) that correspond to the input buffer’s data in order to eliminate stale entries. That done, it is safe for the driver to retrieve the input data from memory (line 21). Remember to free (line 23) the buffer acquired from the memory allocator. The driver will return **OK** (line 24) to distinguish a successful from an unsuccessful operation.

```

STATUS drvExample1 ()          /* simple driver - good performance */
{
3: void *      pInBuf;        /* input buffer */
4: void *      pOutBuf;      /* output buffer */

6: pOutBuf = memalign (_CACHE_ALIGN_SIZE, BUF_SIZE);
8: if (pOutBuf == NULL)
9:     return (ERROR);        /* memory allocator failed */
11: /* other driver initialization and buffer filling */
13: cacheFlush (DATA_CACHE, pOutBuf, BUF_SIZE);
14: drvWrite (pOutBuf);       /* output data to device */
16: /* more driver code */
18: cacheClear (DATA_CACHE, pInBuf, BUF_SIZE);
19: pInBuf = drvRead ();      /* wait for device data */
21: /* handle input data from device */
23: free (pOutBuf);          /* return buffer to memory pool */
24: return (OK);
}

```

Extending this flush/invalidate concept further, individual buffers can be treated this way, not just the entire cache system. The idea is to avoid unnecessary flush and/or invalidate operations on a per-buffer basis by allocating cache-safe buffers. Calls to `cacheDmaMalloc()` optimize the flush and invalidate function pointers to NULL, if possible, while maintaining data integrity. Example 2 The following example is of a high-performance driver that takes advantage of the cache library to maintain coherency. It uses `cacheDmaMalloc()` and the macros `CACHE_DMA_FLUSH` and `CACHE_DMA_INVALIDATE`. A buffer pointer is passed as a parameter (line 2). If the pointer is not NULL (line 7), it is assumed that the buffer will not experience any cache coherency problems. If the driver was not provided with a cache-safe buffer, it will get one (line 11) from `cacheDmaMalloc()`. A `CACHE_FUNCS` structure (see `cacheLib.h`) is used to create a buffer that will not suffer from cache coherency problems. If the memory allocator fails (line 13), the driver will typically return `ERROR` (line 14) and quit.

The driver fills the output buffer with initialization information, device commands, and data (line 17), and is prepared to pass the buffer to the device. Before doing so, the driver must flush the data cache (line 19) to ensure that the buffer is in memory, not hidden in the cache. The routine `drvWrite()` lets the device know that the data is ready and where in memory it is located (line 20).

More driver code is executed (line 22), and the driver is then ready to receive data that the device has placed in the buffer in memory (line 24). Before the driver cache can work with the incoming data, it must invalidate the data cache entries (line 25) that correspond to the input buffer's data in order to eliminate stale entries. That done, it is safe for the driver to handle the input data (line 27), which the driver retrieves from memory. Remember to free the buffer (line 29) acquired from the memory allocator. The driver will return `OK` (line 30) to distinguish a successful from an unsuccessful operation.

**cacheLib**

```

STATUS drvExample2 (pBuf)          /* simple driver - great performance */
2: void *      pBuf;              /* buffer pointer parameter */
    {
5:  if (pBuf != NULL)
        {
7:      /* no cache coherency problems with buffer passed to driver */
        }
    else
        {
11:     pBuf = cacheDmaMalloc (BUF_SIZE);
13:     if (pBuf == NULL)
14:         return (ERROR);      /* memory allocator failed */
        }
17: /* other driver initialization and buffer filling */
19: CACHE_DMA_FLUSH (pBuf, BUF_SIZE);
20: drvWrite (pBuf);              /* output data to device */
22: /* more driver code */
24: drvWait ();                  /* wait for device data */
25: CACHE_DMA_INVALIDATE (pBuf, BUF_SIZE);
27: /* handle input data from device */
29: cacheDmaFree (pBuf);         /* return buffer to memory pool */
30: return (OK);
    }

```

Do not use `CACHE_DMA_FLUSH` or `CACHE_DMA_INVALIDATE` without first calling `cacheDmaMalloc()`, otherwise the function pointers may not be initialized correctly. Note that this driver scheme assumes all cache coherency modes have been set before driver initialization, and that the modes do not change after driver initialization. The `cacheFlush()` and `cacheInvalidate()` functions can be used at any time throughout the system since they are affiliated with the hardware, not the malloc/free buffer.

A call to `cacheLibInit()` in write-through mode makes the flush function pointers NULL. Setting the caches in copyback mode (if supported) should set the pointer to and call an architecture-specific flush routine. The invalidate and flush macros may be NULLified if the hardware provides bus snooping and there are no cache coherency problems. Example 3 The next example shows a more complex driver that requires address translations to assist in the cache coherency scheme. The previous example had *a priori* knowledge of the system memory map and/or the device interaction with the memory system. This next driver demonstrates a case in which the virtual address returned by `cacheDmaMalloc()` might differ from the physical address seen by the device. It uses the `CACHE_DMA_VIRT_TO_PHYS` and `CACHE_DMA_PHYS_TO_VIRT` macros in addition to the `CACHE_DMA_FLUSH` and `CACHE_DMA_INVALIDATE` macros.

The `cacheDmaMalloc()` routine initializes the buffer pointer (line 3). If the memory allocator fails (line 5), the driver will typically return `ERROR` (line 6) and quit. The driver fills the output buffer with initialization information, device commands, and data (line 8), and is prepared to pass the buffer to the device. Before doing so, the driver must flush the

data cache (line 10) to ensure that the buffer is in memory, not hidden in the cache. The flush is based on the virtual address since the processor filled in the buffer. The `drvWrite()` routine lets the device know that the data is ready and where in memory it is located (line 11). Note that the `CACHE_DMA_VIRT_TO_PHYS` macro converts the buffer's virtual address to the corresponding physical address for the device.

More driver code is executed (line 13), and the driver is then ready to receive data that the device has placed in the buffer in memory (line 15). Note the use of the `CACHE_DMA_PHYS_TO_VIRT` macro on the buffer pointer received from the device. Before the driver cache can work with the incoming data, it must invalidate the data cache entries (line 16) that correspond to the input buffer's data in order to eliminate stale entries. That done, it is safe for the driver to handle the input data (line 17), which it retrieves from memory. Remember to free (line 19) the buffer acquired from the memory allocator. The driver will return OK (line 20) to distinguish a successful from an unsuccessful operation.

```
STATUS drvExample3 ()          /* complex driver - great performance */ {
3: void * pBuf = cacheDmaMalloc (BUF_SIZE);
5: if (pBuf == NULL)
6:     return (ERROR);          /* memory allocator failed */
8: /* other driver initialization and buffer filling */
10: CACHE_DMA_FLUSH (pBuf, BUF_SIZE);
11: drvWrite (CACHE_DMA_VIRT_TO_PHYS (pBuf));
13: /* more driver code */
15: pBuf = CACHE_DMA_PHYS_TO_VIRT (drvRead ());
16: CACHE_DMA_INVALIDATE (pBuf, BUF_SIZE);
17: /* handle input data from device */
19: cacheDmaFree (pBuf);        /* return buffer to memory pool */
20: return (OK);
}
```

### Driver Summary

The virtual-to-physical and physical-to-virtual function pointers associated with `cacheDmaMalloc()` are supplements to a cache-safe buffer. Since the processor operates on virtual addresses and the devices access physical addresses, discrepant addresses can occur and might prevent DMA-type devices from being able to access the allocated buffer. Typically, the MMU is used to return a buffer that has pages marked as non-cacheable. An MMU is used to translate virtual addresses into physical addresses, but it is not guaranteed that this will be a “transparent” translation.

When `cacheDmaMalloc()` does something that makes the virtual address different from the physical address needed by the device, it provides the translation procedures. This is often the case when using translation lookaside buffers (TLB) or a segmented address space to inhibit caching (*e.g.*, by creating a different virtual address for the same physical space.) If the virtual address returned by `cacheDmaMalloc()` is the same as the physical address, the function pointers are made NULL so that no calls are made when the macros

**cacheLib**

are expanded. Board Support Packages Each board for an architecture with more than one cache implementation has the potential for a different cache system. Hence the BSP for selecting the appropriate cache library. The function pointer **sysCacheLibInit** is set to **cacheXxxLibInit()** (“Xxx” refers to the chip-specific name of a library or function) so that the function pointers for that cache system will be initialized and the linker will pull in only the desired cache library. Below is an example of **cacheXxxLib** being linked in by **sysLib.c**. For systems without caches and for those architectures with only one cache design, there is no need for the **sysCacheLibInit** variable.

```
FUNCPTR sysCacheLibInit = (FUNCPTR) cacheXxxLibInit;
```

For cache systems with bus snooping, the flush and invalidate macros should be NULLified to enhance system and driver performance in **sysHwInit()**.

```
void sysHwInit ()
{
...
cacheLib.flushRtn = NULL;          /* no flush necessary */
cacheLib.invalidateRtn = NULL;    /* no invalidate necessary */
...
}
```

There may be some drivers that require numerous cache calls, so many that they interfere with the code clarity. Additional checking can be done at the initialization stage to determine if **cacheDmaMalloc()** returned a buffer in non-cacheable space. Remember that it will return a cache-safe buffer by virtue of the function pointers. Ideally, these are NULL, since the MMU was used to mark the pages as non-cacheable. The macros **CACHE\_Xxx\_IS\_WRITE\_COHERENT** and **CACHE\_Xxx\_IS\_READ\_COHERENT** can be used to check the flush and invalidate function pointers, respectively.

Write buffers are used to allow the processor to continue execution while the bus interface unit moves the data to the external device. In theory, the write buffer should be smart enough to flush itself when there is a write to non-cacheable space or a read of an item that is in the buffer. In those cases where the hardware does not support this, the software must flush the buffer manually. This often is accomplished by a read to non-cacheable space or a NOP instruction that serializes the chip’s pipelines and buffers. This is not really a caching issue; however, the cache library provides a **CACHE\_PIPE\_FLUSH** macro. External write buffers may still need to be handled in a board-specific manner.

**INCLUDE FILES**    **cacheLib.h**

**SEE ALSO**        Architecture-specific cache-management libraries (**cacheXxxLib**), **vmLib**, *VxWorks Programmer’s Guide: I/O System*

---

## cacheR3kLib

<b>NAME</b>	<b>cacheR3kLib</b> – MIPS R3000 cache management library
<b>ROUTINES</b>	<b>cacheR3kLibInit()</b> - initialize the R3000 cache library
<b>DESCRIPTION</b>	<p>This library contains architecture-specific cache library functions for the MIPS R3000 architecture. The R3000 utilizes a variable-size instruction and data cache that operates in write-through mode. Cache line size also varies. Cache tags may be invalidated on a per-word basis by execution of a byte write to a specified word while the cache is isolated. See also the manual entry for <b>cacheR3kALib</b>.</p> <p>For general information about caching, see the manual entry for <b>cacheLib</b>.</p>
<b>INCLUDE FILES</b>	<b>cacheLib.h</b>
<b>SEE ALSO</b>	<b>cacheR3kALib</b> , <b>cacheLib</b> , Gerry Kane: <i>MIPS R3000 RISC Architecture</i>

---

## cacheR4kLib

<b>NAME</b>	<b>cacheR4kLib</b> – MIPS R4000 cache management library
<b>ROUTINES</b>	<b>cacheR4kLibInit()</b> - initialize the R4000 cache library
<b>DESCRIPTION</b>	<p>This library contains architecture-specific cache library functions for the MIPS R4000 architecture. The R4000 utilizes a variable-size instruction and data cache that operates in write-back mode. Cache line size also varies.</p> <p>For general information about caching, see the manual entry for <b>cacheLib</b>.</p>
<b>INCLUDE FILES</b>	<b>cacheLib.h</b>
<b>SEE ALSO</b>	<b>cacheLib</b>

## cacheR5kLib

<b>NAME</b>	<b>cacheR5kLib</b> – MIPS R5000 cache management library
<b>ROUTINES</b>	<b>cacheR5kLibInit()</b> - initialize the R5000 cache library
<b>DESCRIPTION</b>	<p>This library contains architecture-specific cache library functions for the MIPS R5000 architecture. The R5000 utilizes a variable-size instruction and data cache that operates in write-back mode. Cache line size also varies.</p> <p>For general information about caching, see the manual entry for <b>cacheLib</b>.</p>
<b>INCLUDE FILES</b>	<b>cacheLib.h</b>
<b>SEE ALSO</b>	<b>cacheLib</b>

---

## cacheR7kLib

<b>NAME</b>	<b>cacheR7kLib</b> – MIPS R7000 cache management library
<b>ROUTINES</b>	<b>cacheR7kLibInit()</b> - initialize the R7000 cache library
<b>DESCRIPTION</b>	<p>This library contains architecture-specific cache library functions for the MIPS R7000 architecture. The R7000 utilizes a variable-size instruction and data cache that operates in write-back mode. Cache line size also varies.</p> <p>For general information about caching, see the manual entry for <b>cacheLib</b>.</p>
<b>INCLUDE FILES</b>	<b>cacheLib.h</b>
<b>SEE ALSO</b>	<b>cacheLib</b>

---

## cacheR10kLib

<b>NAME</b>	<b>cacheR10kLib</b> – MIPS R10000 cache management library
<b>ROUTINES</b>	<b>cacheR10kLibInit()</b> - initialize the R10000 cache library
<b>DESCRIPTION</b>	<p>This library contains architecture-specific cache library functions for the MIPS R10000 architecture. The R10000 utilizes a variable-size instruction and data cache that operates in write-back mode. Cache line size also varies.</p> <p>For general information about caching, see the manual entry for <b>cacheLib</b>.</p>
<b>INCLUDE FILES</b>	<b>cacheLib.h</b>
<b>SEE ALSO</b>	<b>cacheLib</b>

---

## cacheR32kLib

<b>NAME</b>	<b>cacheR32kLib</b> – MIPS RC32364 cache management library
<b>ROUTINES</b>	<b>cacheR32kLibInit()</b> - initialize the RC32364 cache library <b>cacheR32kMalloc()</b> - allocate a cache-safe buffer, if possible
<b>DESCRIPTION</b>	<p>This library contains architecture-specific cache library functions for the MIPS IDT RC32364 architecture.</p> <p>For general information about caching, see the manual entry for <b>cacheLib</b>.</p>
<b>INCLUDE FILES</b>	<b>cacheLib.h</b>
<b>SEE ALSO</b>	<b>cacheLib</b>

## cacheR33kLib

<b>NAME</b>	<b>cacheR33kLib</b> – MIPS R33000 cache management library
<b>ROUTINES</b>	<b>cacheR33kLibInit()</b> - initialize the R33000 cache library
<b>DESCRIPTION</b>	<p>This library contains architecture-specific cache library functions for the MIPS R33000 architecture. The R33000 utilizes a 8-Kbyte instruction cache and a 1-Kbyte data cache that operate in write-through mode. Cache line size is fixed at 16 bytes. Cache tags may be invalidated on a per-line basis by execution of a store to a specified line while the cache is in invalidate mode.</p> <p>For general information about caching, see the manual entry for <b>cacheLib</b>.</p>
<b>INCLUDE FILES</b>	<b>arch/mips/lr33000.h</b> , <b>cacheLib.h</b>
<b>SEE ALSO</b>	<b>cacheLib</b> , <i>LSI Logic LR33000 MIPS Embedded Processor User's Manual</i>

---

## cacheR333x0Lib

<b>NAME</b>	<b>cacheR333x0Lib</b> – MIPS R333x0 cache management library
<b>ROUTINES</b>	<b>cacheR333x0LibInit()</b> - initialize the R333x0 cache library
<b>DESCRIPTION</b>	<p>This library contains architecture-specific cache library functions for the MIPS R333x0 architecture. The R33300 utilizes a 4-Kbyte instruction cache and a 2-Kbyte data cache that operate in write-through mode. The R33310 utilizes a 8-Kbyte instruction cache and a 4-Kbyte data cache that operate in write-through mode. Cache line size is fixed at 16 bytes. Cache tags may be invalidated on a per-line basis by execution of a store to a specified line while the cache is in invalidate mode.</p> <p>For general information about caching, see the manual entry for <b>cacheLib</b>.</p>
<b>INCLUDE FILES</b>	<b>arch/mips/lr33300.h</b> , <b>cacheLib.h</b>
<b>SEE ALSO</b>	<b>cacheLib</b> , <i>LSI Logic LR33300 and LR33310 Self-Embedding Processors User's Manual</i>

---

## cacheSh7040Lib

<b>NAME</b>	<b>cacheSh7040Lib</b> – Hitachi SH7040 cache management library
<b>ROUTINES</b>	<b>cacheSh7040LibInit()</b> - initialize the SH7040 cache library
<b>DESCRIPTION</b>	This library contains architecture-specific cache library functions for the Hitachi SH7040 architecture. This architecture has a 1-Kbyte instruction cache. For general information about caching, see the manual entry for <b>cacheLib</b> .
<b>INCLUDE FILES</b>	<b>cacheLib.h</b>
<b>SEE ALSO</b>	<b>cacheLib</b>

---

## cacheSh7604Lib

<b>NAME</b>	<b>cacheSh7604Lib</b> – Hitachi SH7604/SH7615 cache management library
<b>ROUTINES</b>	<b>cacheSh7604LibInit()</b> - initialize the SH7604/SH7615 cache library
<b>DESCRIPTION</b>	This library contains architecture-specific cache library functions for the Hitachi SH7604/SH7615 instruction and data mixed cache.
<b>INCLUDE FILES</b>	<b>cacheLib.h</b>
<b>SEE ALSO</b>	<b>cacheLib</b>

## cacheSh7622Lib

<b>NAME</b>	<b>cacheSh7622Lib</b> – SH7622 cache management library
<b>ROUTINES</b>	<b>cacheSh7622LibInit()</b> - initialize the SH7622 cache library
<b>DESCRIPTION</b>	This library contains architecture-specific cache library functions for the Hitachi SH7622 instruction and data caches.
<b>INCLUDE FILES</b>	<b>cacheLib.h</b>
<b>SEE ALSO</b>	<b>cacheLib</b>

---

## cacheSh7700Lib

<b>NAME</b>	<b>cacheSh7700Lib</b> – Hitachi SH7700 cache management library
<b>ROUTINES</b>	<b>cacheSh7700LibInit()</b> - initialize the SH7700 cache library
<b>DESCRIPTION</b>	<p>This library contains architecture-specific cache library functions for the Hitachi SH7700 architecture. There is a 8-Kbyte (2-Kbyte for SH7702) mixed instruction and data cache that operates in write-through or write-back (copyback) mode. The 8-Kbyte cache can be divided into 4-Kbyte cache and 4-Kbyte memory. Cache line size is fixed at 16 bytes, and the cache address array holds physical addresses as cache tags. Cache entries may be “flushed” by accesses to the address array in privileged mode. There is a write-back buffer which can hold one line of cache entry, and the completion of write-back cycle is assured by accessing to any cache through region.</p> <p>For general information about caching, see the manual entry for <b>cacheLib</b>.</p>
<b>INCLUDE FILES</b>	<b>cacheLib.h</b>
<b>SEE ALSO</b>	<b>cacheLib</b>

---

## cacheSh7729Lib

<b>NAME</b>	<b>cacheSh7729Lib</b> – Hitachi SH7729 cache management library
<b>ROUTINES</b>	<b>cacheSh7729LibInit()</b> - initialize the SH7729 cache library
<b>DESCRIPTION</b>	<p>This library contains architecture-specific cache library functions for the Hitachi SH7729 architecture.</p> <p>The cache is 16-Kbytes (16 bytes X 256 entries X 4 ways) mixed instruction and data cache that operates in write-through or write-back (copyback) mode. Cache line size is fixed at 16 bytes, and the cache address array holds physical addresses as cache tags. Cache entries may be “flushed” by accesses to the address array in privileged mode. There is a write-back buffer which can hold one line of cache entry, and the completion of write-back cycle is assured by accessing to any cache through region.</p> <p>For general information about caching, see the manual entry for <b>cacheLib</b>.</p>
<b>INCLUDE FILES</b>	<b>cacheLib.h</b>
<b>SEE ALSO</b>	<b>cacheLib</b>

---

## cacheSh7750Lib

<b>NAME</b>	<b>cacheSh7750Lib</b> – Hitachi SH7750 cache management library
<b>ROUTINES</b>	<b>cacheSh7750LibInit()</b> - initialize the SH7750 cache library
<b>DESCRIPTION</b>	<p>This library contains architecture-specific cache library functions for the Hitachi SH7750 architecture. There is a 8-Kbyte instruction cache and 16-Kbyte operand cache that operates in write-through or write-back (copyback) mode. The 16-Kbyte operand cache can be divided into 8-Kbyte cache and 8-Kbyte memory. Cache line size is fixed at 32 bytes, and the cache address array holds physical addresses as cache tags. Cache entries may be “flushed” by accesses to the address array in privileged mode. There is a write-back buffer which can hold one line of cache entry, and the completion of write-back cycle is assured by accessing to any cache through region.</p> <p>For general information about caching, see the manual entry for <b>cacheLib</b>.</p>
<b>INCLUDE FILES</b>	<b>cacheLib.h</b>
<b>SEE ALSO</b>	<b>cacheLib</b>

## cacheSun4Lib

<b>NAME</b>	<b>cacheSun4Lib</b> – Sun-4 cache management library
<b>ROUTINES</b>	<b>cacheSun4LibInit()</b> - initialize the Sun-4 cache library <b>cacheSun4ClearLine()</b> - clear a line from a Sun-4 cache <b>cacheSun4ClearPage()</b> - clear a page from a Sun-4 cache <b>cacheSun4ClearSegment()</b> - clear a segment from a Sun-4 cache <b>cacheSun4ClearContext()</b> - clear a specific context from a Sun-4 cache
<b>DESCRIPTION</b>	<p>This library contains architecture-specific cache library functions for the Sun Microsystems Sun-4 architecture. There is a 64-Kbyte mixed instruction and data cache that operates in write-through mode. Each cache line contains 16 bytes. Cache tags may be “flushed” by accesses to alternate space in supervisor mode. Invalidate operations are performed in software by writing zero to the cache tags in an iterative manner. Tag operations are performed on “page,” “segment,” or “context” granularity.</p> <p>MMU (Memory Management Unit) support is needed to mark pages cacheable or non-cacheable. For more information, see the manual entry for <b>vmLib</b>.</p> <p>For general information about caching, see the manual entry for <b>cacheLib</b>.</p>
<b>INCLUDE FILES</b>	<b>cacheLib.h</b>
<b>SEE ALSO</b>	<b>cacheLib</b> , <b>vmLib</b>

---

## cacheTx49Lib

<b>NAME</b>	<b>cacheTx49Lib</b> – Toshiba Tx49 cache management library
<b>ROUTINES</b>	<b>cacheTx49LibInit()</b> - initialize the Tx49 cache library
<b>DESCRIPTION</b>	<p>This library contains architecture-specific cache library functions for the Toshiba Tx49 architecture. The Tx49 utilizes a variable-size instruction and data cache that operates in write-back mode. The cache is four-way set associative and the library allows the cache line size to vary.</p> <p>For general information about caching, see the manual entry for <b>cacheLib</b>.</p>
<b>INCLUDE FILES</b>	<b>cacheLib.h</b>
<b>SEE ALSO</b>	<b>cacheLib</b>

---

## cbioLib

<b>NAME</b>	<b>cbioLib</b> – cached block I/O library
<b>ROUTINES</b>	<b>cbioLibInit()</b> - Initialize CBIO Library <b>cbioBlkRW()</b> - transfer blocks to or from memory <b>cbioBytesRW()</b> - transfer bytes to or from memory <b>cbioBlkCopy()</b> - block to block (sector to sector) transfer routine <b>cbioIoctl()</b> - perform ioctl operation on device <b>cbioModeGet()</b> - return the mode setting for CBIO device <b>cbioModeSet()</b> - set mode for CBIO device <b>cbioRdyChgdGet()</b> - determine ready status of CBIO device <b>cbioRdyChgdSet()</b> - force a change in ready status of CBIO device <b>cbioLock()</b> - obtain CBIO device semaphore. <b>cbioUnlock()</b> - release CBIO device semaphore. <b>cbioParamsGet()</b> - fill in <b>CBIO_PARAMS</b> structure with CBIO device parameters <b>cbioShow()</b> - print information about a CBIO device <b>cbioDevVerify()</b> - verify <b>CBIO_DEV_ID</b> <b>cbioWrapBlkDev()</b> - create CBIO wrapper atop a <b>BLK_DEV</b> device <b>cbioDevCreate()</b> - Initialize a CBIO device (Generic)
<b>DESCRIPTION</b>	<p>This library provides the Cached Block Input Output Application Programmers Interface (CBIO API). Libraries such as <b>dosFsLib</b>, <b>rawFsLib</b>, and <b>usrFdiskPartLib</b> use the CBIO API for I/O operations to underlying devices.</p> <p>This library also provides generic services for CBIO modules. The libraries <b>dpartCbio</b>, <b>dcacheCbio</b>, and <b>ramDiskCbio</b> are examples of CBIO modules that make use of these generic services.</p> <p>This library also provides a CBIO module that converts <b>blkIo</b> driver <b>BLK_DEV</b> (<b>blkIo.h</b>) interface into CBIO API compliant interface using minimal memory overhead. This lean module is known as the basic <b>BLK_DEV</b> to CBIO wrapper module.</p>
<b>CBIO MODULES AND DEVICES</b>	<p>A CBIO module contains code for supporting CBIO devices. The libraries <b>cbioLib</b>, <b>dcacheCbio</b>, <b>dpartCbio</b>, and <b>ramDiskCbio</b> are examples of CBIO modules.</p> <p>A CBIO device is a software layer that provide its master control of I/O to it subordinate. CBIO device layers typically reside logically below a file system and above a storage device. CBIO devices conform to the CBIO API on their master (upper) interface.</p> <p>CBIO modules provide a CBIO device creation routine used to instantiate a CBIO device. The CBIO modules device creation routine returns a <b>CBIO_DEV_ID</b> handle. The <b>CBIO_DEV_ID</b> handle is used to uniquely identify the CBIO device layer instance. The user of the CBIO device passes this handle to the CBIO API routines when accessing the device.</p>

The libraries **dosFsLib**, **rawFsLib**, and **usrFdiskPartLib** are considered users of CBIO devices because they use the CBIO API on their subordinate (lower) interface. They do not conform to the CBIO API on their master interface, therefore they are not CBIO modules. They are users of CBIO devices and always reside above CBIO devices in the logical stack.

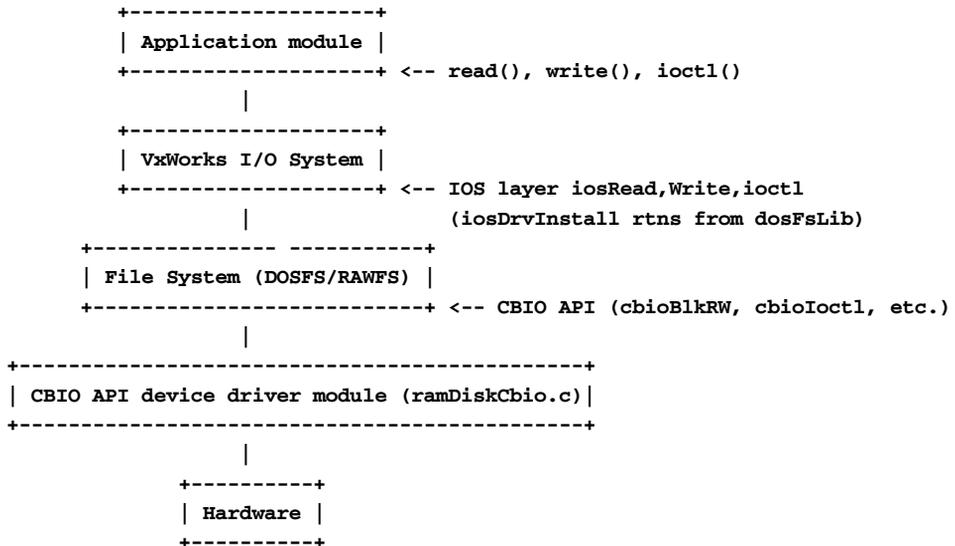
#### TYPES OF CBIO DEVICES

A “CBIO to CBIO device” uses the CBIO API for both its master and its subordinate interface. Typically, some type of module specific I/O processing occurs during the interface between the master and subordinate layers. The libraries **dpartCbio** and **dcacheCbio** are examples of CBIO to CBIO devices. CBIO to CBIO device layers are stackable. Care should be taken to assemble the stack properly. Refer to each modules reference manual entry for recommendations about the optimum stacking order.

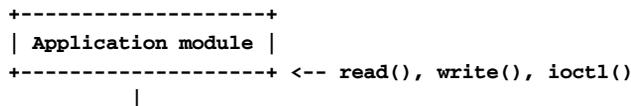
A “CBIO API device driver” is a device driver which provides the CBIO API as the interface between the hardware and its upper layer. The **ramDiskCbio.c** RAM DISK driver is an example of a simple CBIO API device driver.

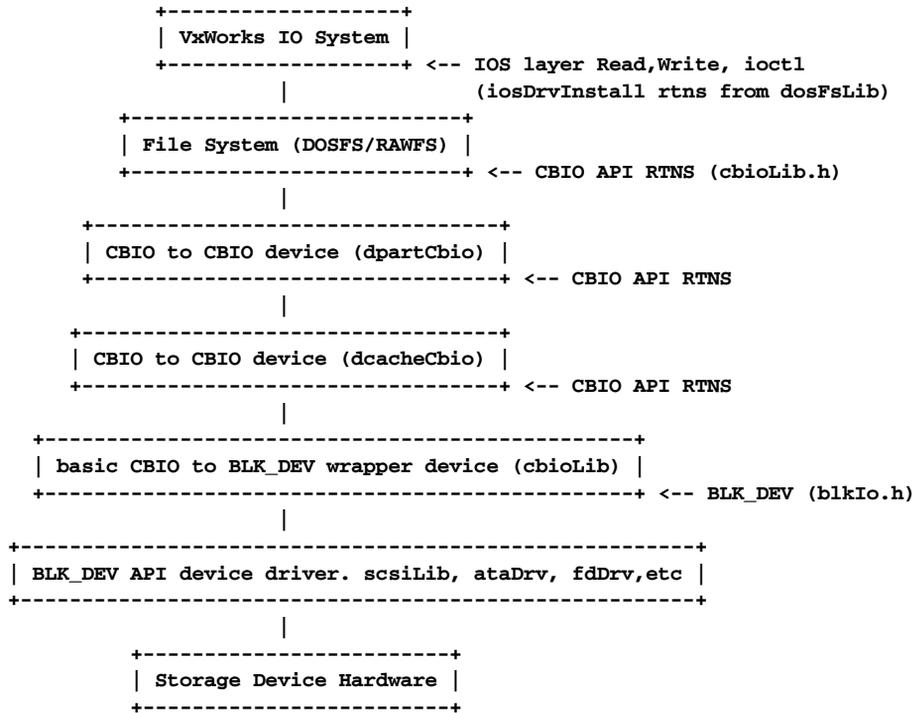
A “basic BLK\_DEV to CBIO wrapper device” wraps a subordinate BLK\_DEV layer with a CBIO API compatible layer. The wrapper is provided via **cbioWrapBlkDev()**.

The logical layers of a typical system using a CBIO RAM DISK are as pictured below:



The logical layers of a typical system with a fixed disk using CBIO partitioning layer and a CBIO caching layer appears:





**PUBLIC CBIO API** The CBIO API provides user access to CBIO devices. Users of CBIO devices are typically either file systems or other CBIO devices.

The CBIO API is exposed via **cbioLib.h**. Users of CBIO modules include the **cbioLib.h** header file. The libraries **dosFsLib**, **dosFsFat**, **dosVDirLib**, **dosDirOldLib**, **usrFdiskPartLib**, and **rawFsLib** all use the CBIO API to access CBIO modules beneath them.

The following functions make up the public CBIO API:

**cbioLibInit()** - Library initialization routine

**cbioBlkRW()** - Transfer blocks (sectors) from/to a memory buffer

**cbioBytesRW()** - Transfer bytes from/to a memory buffer

**cbioBlkCopy()** - Copy directly from block to block (sector to sector)

**cbioIoctl()** - Perform I/O control operations on the CBIO device

**cbioModeGet()** - Get the CBIO device mode (**O\_RDONLY**, **O\_WRONLY**, or **O\_RDWR**)

**cbioModeSet()** - Set the CBIO device mode (**O\_RDONLY**, **O\_WRONLY**, or **O\_RDWR**)

**cbioRdyChgdGet()** - Determine the CBIO device ready status state

**cbioRdyChgdSet()** - Force a change in the CBIO device ready status state

**cbioLock()** - Obtain exclusive ownership of the CBIO device

**cbioUnlock()** - Release exclusive ownership of the CBIO device

**cbioParamsGet()** - Fill a **CBIO\_PARAMS** structure with data from the CBIO device

**cbioDevVerify()** - Verify valid CBIO device

**cbioWrapBlkDev()** - Create CBIO wrapper atop a **BLK\_DEV**

**cbioShow()** - Display information about a CBIO device

These CBIO API functions (except **cbioLibInit()**) are passed a **CBIO\_DEV\_ID** handle in the first argument. This handle (obtained from the subordinate CBIO modules device creation routine) is used by the routine to verify that the CBIO device is valid and then to perform the requested operation on the specific CBIO device.

When the **CBIO\_DEV\_ID** passed to the CBIO API routine is not a valid CBIO handle, **ERROR** will be returned with the **errno** set to **S\_cbioLib\_INVALID\_CBIO\_DEV\_ID** (**cbioLib.h**).

Refer to the individual manual entries for each function for a complete description.

#### THE BASIC CBIO TO **BLK\_DEV** WRAPPER MODULE

The basic CBIO to **BLK\_DEV** wrapper is a minimized disk cache using simplified algorithms. It is used to convert a legacy **BLK\_DEV** device into as CBIO device. It may be used standalone with solid state disks which do not have mechanical seek and rotational latency delays, such flash cards. It may also be used in conjunction with the **dpartCbio** and **dcacheCbio** libraries. The DOS file system **dosFsDevCreate()** routine will call **cbioWrapBlkDev()** internally, so the file system may be installed directly on top of a block driver **BLK\_DEV** or it can be used with cache and partitioning support.

The function **cbioWrapBlkDev()** is used to create the CBIO wrapper atop a **BLK\_DEV** device.

The functions **dcacheDevCreate()** and **dpartDevCreate()** also both internally use **cbioDevVerify()** and **cbioWrapBlkDev()** to either stack the new CBIO device atop a validated CBIO device or to create a basic CBIO to **BLK\_DEV** wrapper as needed. The user typically never needs to manually invoke the **cbioWrapBlkDev()** or **cbioDevVerify()** functions.

Please note that the basic CBIO **BLK\_DEV** wrapper is inappropriate for rotational media without the disk caching layer. The services provided by the **dcacheCbio** module are more appropriate for use on rotational disk devices and will yield superior performance when used.

**INCLUDE FILES**     **cbioLib.h, cbioLibP.h**

**SEE ALSO**         *VxWorks Programmers Guide: I/O System.*

---

## cdromFsLib

**NAME** **cdromFsLib** – ISO 9660 CD-ROM read-only file system library

**ROUTINES** **cdromFsInit()** - initialize **cdromFsLib**  
**cdromFsVolConfigShow()** - show the volume configuration information  
**cdromFsDevCreate()** - create a **cdromFsLib** device

**DESCRIPTION** This library defines **cdromFsLib**, a utility that lets you use standard POSIX I/O calls to read data from a CD-ROM formatted according to the ISO 9660 standard file system.

It provides access to CD-ROM file systems using any standard **BLOCK\_DEV** structure (that is, a disk-type driver).

The basic initialization sequence is similar to installing a DOS file system on a SCSI device.

1. Initialize the cdrom file system library (preferably in **sysScsiConfig()** in **sysScsi.c**):

```
cdromFsInit ();
```

2. Locate and create a SCSI physical device:

```
pPhysDev=scsiPhysDevCreate(pSysScsiCtrl,0,0,0,NONE,1,0,0);
```

3. Create a SCSI block device on the physical device:

```
pBlkDev = (SCSI_BLK_DEV *) scsiBlkDevCreate (pPhysDev, 0, 0);
```

4. Create a CD-ROM file system on the block device:

```
cdVolDesc = cdromFsDevCreate ("cdrom:", (BLK_DEV *) pBlkDev);
```

Call **cdromFsDevCreate()** once for each CD-ROM drive attached to your target. After the successful completion of **cdromFsDevCreate()**, the CD-ROM file system will be available like any DOS file system, and you can access data on the named CD-ROM device using **open()**, **close()**, **read()**, **ioctl()**, **readdir()**, and **stat()**. A **write()** always returns an error.

The **cdromFsLib** utility supports multiple drives, concurrent access from multiple tasks, and multiple open files.

### FILE AND DIRECTORY NAMING

The strict ISO 9660 specification allows only uppercase file names consisting of 8 characters plus a 3 character suffix. To support multiple versions of the same file, the ISO 9660 specification also supports version numbers. When specifying a file name in an **open()** call, you can select the file version by appending the file name with a semicolon (;) followed by a decimal number indicating the file version. If you omit the version number, **cdromFsLib** opens the latest version of the file.

To accommodate users familiar with MS-DOS, **cdromFsLib** lets you use lowercase name arguments to access files with names consisting entirely of uppercase characters. Mixed-case file and directory names are accessible only if you specify their exact case-correct names.

For the time being, **cdromFsLib** further accommodates MS-DOS users by allowing “\” (backslash) instead of “/” in path names. However, the use of the backslash is discouraged because it may not be supported in future versions of **cdromFsLib**.

Finally, **cdromFsLib** uses an 8-bit clean implementation of ISO 9660. Thus, **cdromFsLib** is compatible with CD-ROMs using either Latin or Asian characters in the file names.

#### **IOCTL CODES SUPPORTED**

##### **FIOGETNAME**

Returns the file name for a specific file descriptor.

##### **FIOLABELGET**

Retrieves the volume label. This code can be used to verify that a particular volume has been inserted into the drive.

##### **FIOWHERE**

Determines the current file position.

##### **FIOSEEK**

Changes the current file position.

##### **FIONREAD**

Tells you the number of bytes between the current location and the end of this file.

##### **FIOREADDIR**

Reads the next directory entry.

##### **FIODISKCHANGE**

Announces that a disk has been replaced (in case the block driver is not able to provide this indication).

##### **FIOUNMOUNT**

Announces that the a disk has been removed (all currently open file descriptors are invalidated).

##### **FIOFSTATGET**

Gets the file status information (directory entry data).

#### **MODIFYING A BSP TO USE CDROMFS**

The following example describes mounting cdromFS on a SCSI device.

Edit your BSP's **config.h** to make the following changes:

1. Insert the following macro definition:

```
#define INCLUDE_CDROMFS
```

2. Change `FALSE` to `TRUE` in the section under the following comment:

```
/* change FALSE to TRUE for SCSI interface */
```

Make the following changes in `sysScsi.c` (or `sysLib.c` if your BSP has no `sysScsi.c`):

The main goal of the above code fragment is to call `cdromFsDevCreate()`. As input, `cdromFsDevCreate()` expects a pointer to a block device. In the example above, the `scsiPhysDevCreate()` and `scsiBlkDevCreate()` calls set up a block device interface for a SCSI CD-ROM device.

After the successful completion of `cdromFsDevCreate()`, the device called “cdrom” is accessible using the standard `open()`, `close()`, `read()`, `ioctl()`, `readdir()`, and `stat()` calls.

**INCLUDE FILES** `cdromFsLib.h`

**CAVEATS** The `cdromFsLib` utility does not support CD sets containing multiple disks.

**SEE ALSO** `ioLib`, ISO 9660 Specification

---

## clockLib

**NAME** `clockLib` – clock library (POSIX)

**ROUTINES** `clock_getres()` - get the clock resolution (POSIX)  
`clock_setres()` - set the clock resolution  
`clock_gettime()` - get the current time of the clock (POSIX)  
`clock_settime()` - set the clock to a specified time (POSIX)

**DESCRIPTION** This library provides a clock interface, as defined in the IEEE standard, POSIX 1003.1b.

A clock is a software construct that keeps time in seconds and nanoseconds. The clock has a simple interface with three routines: `clock_settime()`, `clock_gettime()`, and `clock_getres()`. The non-POSIX routine `clock_setres()` that was provided so that `clockLib` could be informed if there were changes in the system clock rate is no longer necessary. This routine is still present for backward compatibility, but does nothing.

Times used in these routines are stored in the `timespec` structure:

```
struct timespec
{
    time_t      tv_sec;          /* seconds */
    long       tv_nsec;        /* nanoseconds (0 -1,000,000,000) */
};
```

- IMPLEMENTATION** Only one *clock\_id* is supported, the required **CLOCK\_REALTIME**. Conceivably, additional “virtual” clocks could be supported, or support for additional auxiliary clock hardware (if available) could be added.
- INCLUDE FILES** **timers.h**
- SEE ALSO** IEEE *VxWorks Programmer’s Guide: Basic OS*, POSIX 1003.1b documentation

---

## cplusLib

- NAME** **cplusLib** – basic run-time support for C++
- ROUTINES**
- cplusCallNewHandler()** - call the allocation failure handler (C++)
  - cplusCtors()** - call static constructors (C++)
  - cplusCtorsLink()** - call all linked static constructors (C++)
  - cplusDemanglerSet()** - change C++ demangling mode (C++)
  - cplusDemanglerStyleSet()** - change C++ demangling style (C++)
  - cplusDtors()** - call static destructors (C++)
  - cplusDtorsLink()** - call all linked static destructors (C++)
  - cplusLibInit()** - initialize the C++ library (C++)
  - cplusXtorSet()** - change C++ static constructor calling strategy (C++)
  - operator delete()** - default run-time support for memory deallocation (C++)
  - operator new()** - default run-time support for operator new (C++)
  - operator new()** - default run-time support for operator new (nothrow) (C++)
  - operator new()** - run-time support for operator new with placement (C++)
  - set\_new\_handler()** - set **new\_handler** to user-defined function (C++)
  - set\_terminate()** - set terminate to user-defined function (C++)
- DESCRIPTION** This library provides run-time support and shell utilities that support the development of VxWorks applications in C++. The run-time support can be broken into three categories:
- Support for C++ new and delete operators.
  - Support for initialization and cleanup of static objects.
- Shell utilities are provided for:
- Resolving overloaded C++ function names.
  - Hiding C++ name mangling, with support for terse or complete name demangling.
  - Manual or automatic invocation of static constructors and destructors.
- The usage of **cplusLib** is more fully described in the *VxWorks Programmer’s Guide: C++ Development*.
- SEE ALSO** *VxWorks Programmer’s Guide: C++ Development*

---

## dbgArchLib

<b>NAME</b>	<b>dbgArchLib</b> – architecture-dependent debugger library
<b>ROUTINES</b>	<p><b>a0()</b> - return the contents of register <b>a0</b> (also <b>a1 - a7</b>) (68K) <b>d0()</b> - return the contents of register <b>d0</b> (also <b>d1 - d7</b>) (68K) <b>sr()</b> - return the contents of the status register (68K, SH) <b>dbgBpTypeBind()</b> - bind a breakpoint handler to a breakpoint type (MIPS) <b>edi()</b> - return the contents of register <b>edi</b> (also <b>esi - eax</b>) (x86) <b>eflags()</b> - return the contents of the status register (x86) <b>r0()</b> - return the contents of register <b>r0</b> (also <b>r1 - r14</b>) (ARM) <b>cpsr()</b> - return the contents of the current processor status register (ARM) <b>psrShow;1()</b> - display the meaning of a specified PSR value, symbolically (ARM) <b>r0()</b> - return the contents of general register <b>r0</b> (also <b>r1-r15</b>) (SH) <b>sr()</b> - return the contents of control register <b>sr</b> (also <b>gbr, vbr</b>) (SH) <b>mach()</b> - return the contents of system register <b>mach</b> (also <b>macl, pr</b>) (SH) <b>o0()</b> - return the contents of register <b>o0</b> (also <b>o1-o7</b>) (SimSolaris) <b>l0()</b> - return the contents of register <b>l0</b> (also <b>l1-l7</b>) (SimSolaris) <b>i0()</b> - return the contents of register <b>i0</b> (also <b>i1-i7</b>) (SimSolaris) <b>npc()</b> - return the contents of the next program counter (SimSolaris) <b>psr()</b> - return the contents of the processor status register (SimSolaris) <b>wim()</b> - return the contents of the window invalid mask register (SimSolaris) <b>y()</b> - return the contents of the y register (SimSolaris) <b>edi()</b> - return the contents of register <b>edi</b> (also <b>esi - eax</b>) (x86/SimNT) <b>eflags()</b> - return the contents of the status register (x86/SimNT)</p>
<b>DESCRIPTION</b>	<p>This module provides architecture-specific support functions for <b>dbgLib</b>. It also includes user-callable functions for accessing the contents of registers in a task's TCB (task control block). These routines include:</p> <p><b>MC680x0:</b></p> <ul style="list-style-type: none"><li><b>a0() - a7()</b> - address registers (<b>a0 - a7</b>)</li><li><b>d0() - d7()</b> - data registers (<b>d0 - d7</b>)</li><li><b>sr()</b> - status register (<b>sr</b>)</li></ul> <p><b>MIPS:</b></p> <ul style="list-style-type: none"><li><b>dbgBpTypeBind()</b> - bind a breakpoint handler to a breakpoint type</li></ul> <p><b>x86/SimNT:</b></p> <ul style="list-style-type: none"><li><b>edi() - eax()</b> - named register values</li><li><b>eflags()</b> - status register value</li></ul>

**SH:**

**r0()** - **r15()** - general registers (**r0** - **r15**)  
**sr()** - status register (**sr**)  
**gbr()** - global base register (**gbr**)  
**vbr()** - vector base register (**vbr**)  
**mach()** - multiply and accumulate register high (**mach**)  
**macl()** - multiply and accumulate register low (**macl**)  
**pr()** - procedure register (**pr**)

**ARM:**

**r0()** - **r14()** - general-purpose registers (**r0** - **r14**)  
**cpsr()** - current processor status reg (**cpsr**)  
**psrShow()** - **psr** value, symbolically

**SimSolaris:**

**g0()** - **g7()** - global registers (**g0** - **g7**)  
**o0()** - **o7()** - out registers (**o0** - **o7**, note lower-case "o")  
**l0()** - **l7()** - local registers (**l0** - **l7**, note lower-case "l")  
**i0()** - **i7()** - in registers (**i0** - **i7**)  
**npc()** - next program counter (**npc**)  
**psr()** - processor status register (**psr**)  
**wim()** - window invalid mask (**wim**)  
**y()** - y register

---

**NOTE:** The routine **pct()**, for accessing the program counter, is found in **usrLib**.

---

**SEE ALSO**

**dbgLib**, *VxWorks Programmer's Guide: Target Shell*

---

## dbgLib

<b>NAME</b>	<b>dbgLib</b> – debugging facilities
<b>ROUTINES</b>	<b>dbgHelp()</b> - display debugging help menu <b>dbgInit()</b> - initialize the local debugging package <b>b()</b> - set or display breakpoints <b>e()</b> - set or display eventpoints (WindView) <b>bh()</b> - set a hardware breakpoint <b>bd()</b> - delete a breakpoint <b>bdall()</b> - delete all breakpoints <b>c()</b> - continue from a breakpoint <b>cret()</b> - continue until the current subroutine returns <b>s()</b> - single-step a task <b>so()</b> - single-step, but step over a subroutine <b>l()</b> - disassemble and display a specified number of instructions <b>tt()</b> - display a stack trace of a task
<b>DESCRIPTION</b>	<p>This library contains VxWorks’s primary interactive debugging routines, which provide the following facilities:</p> <ul style="list-style-type: none"><li>- task breakpoints</li><li>- task single-stepping</li><li>- symbolic disassembly</li><li>- symbolic task stack tracing</li></ul> <p>In addition, <b>dbgLib</b> provides the facilities necessary for enhanced use of other VxWorks functions, including:</p> <ul style="list-style-type: none"><li>- enhanced shell abort and exception handling (via <b>tyLib</b> and <b>excLib</b>)</li></ul> <p>The facilities of <b>excLib</b> are used by <b>dbgLib</b> to support breakpoints, single-stepping, and additional exception handling functions.</p>
<b>INITIALIZATION</b>	<p>The debugging facilities provided by this module are optional. In the standard VxWorks development configuration as distributed, the debugging package is included. The configuration macro is <b>INCLUDE_DEBUG</b>. When defined, it enables the call to <b>dbgInit()</b> in the task <b>usrRoot()</b> in <b>usrConfig.c</b>. The <b>dbgInit()</b> routine initializes <b>dbgLib</b> and must be made before any other routines in the module are called.</p>
<b>BREAKPOINTS</b>	<p>Use the routine <b>b()</b> or <b>bh()</b> to set breakpoints. Breakpoints can be set to be hit by a specific task or all tasks. Multiple breakpoints for different tasks can be set at the same address. Clear breakpoints with <b>bd()</b> and <b>bdall()</b>.</p> <p>When a task hits a breakpoint, the task is suspended and a message is displayed on the console. At this point, the task can be examined, traced, deleted, its variables changed, <i>etc.</i></p>

If you examine the task at this point (using the **i()** routine), you will see that it is in a suspended state. The instruction at the breakpoint address has not yet been executed.

To continue executing the task, use the **c()** routine. The breakpoint remains until it is explicitly removed.

#### EVENTPOINTS (WINDVIEW)

When WindView is installed, **dbgLib** supports eventpoints. Use the routine **e()** to set eventpoints. Eventpoints can be set to be hit by a specific task or all tasks. Multiple eventpoints for different tasks can be set at the same address.

When a task hits an eventpoint, an event is logged and is displayed by VxWorks kernel instrumentation.

You can manage eventpoints with the same facilities that manage breakpoints: for example, unbreakable tasks (discussed below) ignore eventpoints, and the **b()** command (without arguments) displays eventpoints as well as breakpoints. As with breakpoints, you can clear eventpoints with **bd()** and **bdall()**.

#### UNBREAKABLE TASKS

An *unbreakable* task ignores all breakpoints. Tasks can be spawned unbreakable by specifying the task option **VX\_UNBREAKABLE**. Tasks can subsequently be set unbreakable or breakable by resetting **VX\_UNBREAKABLE** with **taskOptionsSet()**. Several VxWorks tasks are spawned unbreakable, such as the shell, the exception support task **excTask()**, and several network-related tasks.

#### DISASSEMBLER AND STACK TRACER

The **I()** routine provides a symbolic disassembler. The **tt()** routine provides a symbolic stack tracer.

#### SHELL ABORT AND EXCEPTION HANDLING

This package includes enhanced support for the shell in a debugging environment. The terminal abort function, which restarts the shell, is invoked with the abort key if the **OPT\_ABORT** option has been set. By default, the abort key is CTRL-C. For more information, see the manual entries for **tyAbortSet()** and **tyAbortFuncSet()**.

#### THE DEFAULT TASK AND TASK REFERENCING

Many routines in this module take an optional task name or ID as an argument. If this argument is omitted or zero, the “current” task is used. The current task (or “default” task) is the last task referenced. The **dbgLib** library uses **taskIdDefault()** to set and get the last-referenced task ID, as do many other VxWorks routines.

All VxWorks shell expressions can reference a task by either ID or name. The shell attempts to resolve a task argument to a task ID; if no match is found in the system symbol table, it searches for the argument in the list of active tasks. When it finds a match, it substitutes the task name with its matching task ID. In symbol lookup, symbol names take precedence over task names.

---

**WARNING:** When a task is continued, `c()` and `s()` routines do not yet distinguish between a suspended task or a task suspended by the debugger. Therefore, use of these routines should be restricted to only those tasks being debugged.

---

**INCLUDE FILES** `dbgLib.h`

**SEE ALSO** `excLib`, `tyLib`, `taskIdDefault()`, `taskOptionsSet()`, `tyAbortSet()`, `tyAbortFuncSet()`, *VxWorks Programmer's Guide: Target Shell*, `windsh`, *Tornado User's Guide: Shell*

---

## dcacheCbio

**NAME** `dcacheCbio` – disk cache driver

**ROUTINES**

- `dcacheDevCreate()` - Create a disk cache
- `dcacheDevDisable()` - Disable the disk cache for this device
- `dcacheDevEnable()` - Re-enable the disk cache
- `dcacheDevTune()` - modify tunable disk cache parameters
- `dcacheDevMemResize()` - set a new size to a disk cache device
- `dcacheShow()` - print information about disk cache
- `dcacheHashTest()` - test hash table integrity

**DESCRIPTION** This module implements a disk cache mechanism via the CBIO API. This is intended for use by the VxWorks DOS file system, to store frequently used disk blocks in memory. The disk cache is unaware of the particular file system format on the disk, and handles the disk as a collection of blocks of a fixed size, typically the sector size of 512 bytes.

The disk cache may be used with SCSI, IDE, ATA, Floppy or any other type of disk controllers. The underlying device driver may be either comply with the CBIO API or with the older block device API.

This library interfaces to device drivers implementing the block device API via the basic CBIO `BLK_DEV` wrapper provided by `cbioLib`.

Because the disk cache complies with the CBIO programming interface on both its upper and lower layers, it is both an optional and a stackable module. It can be used or omitted depending on resources available and performance required.

The disk cache module implements the CBIO API, which is used by the file system module to access the disk blocks, or to access bytes within a particular disk block. This allows the file system to use the disk cache to store file data as well as Directory and File Allocation Table blocks, on a Most Recently Used basis, thus keeping a controllable subset of these disk structures in memory. This results in minimized memory requirements for the file system, while avoiding any significant performance degradation.

The size of the disk cache, and thus the memory consumption of the disk subsystem, is configured at the time of initialization (see **dcacheDevCreate()**), allowing the user to trade-off memory consumption versus performance. Additional performance tuning capabilities are available through **dcacheDevTune()**.

Briefly, here are the main techniques deployed by the disk cache:

- Least Recently Used block re-use policy
- Read-ahead
- Write-behind with sorting and grouping
- Hidden writes
- Disk cache bypass for large requests
- Background disk updating (flushing changes to disk) with an adjustable update period (ioctl flushes occur without delay.)

Some of these techniques are discussed in more detail below; others are described in various professional and academic publications.

#### **DISK CACHE ALGORITHM**

The disk cache is composed internally of a number cache blocks, of the same size as the disk physical block (sector). These cache blocks are maintained in a list in “Most Recently Used” order, that is, blocks which are used are moved to the top of this list. When a block needs to be relinquished, and made available to contain a new disk block, the Least Recently Used block will be used for this purpose.

In addition to the regular cache blocks, some of the memory allocated for cache is set aside for a “big buffer”, which may range from 1/4 of the overall cache size up to 64KB. This buffer is used for:

- Combining cache blocks with adjacent disk block numbers, in order to write them to disk in groups, and save on latency and overhead
- Reading ahead a group of blocks, and then converting them to normal cache blocks.

Because there is significant overhead involved in accessing the disk drive, read-ahead improves performance significantly by reading groups of blocks at once.

#### **TUNABLE PARAMETERS**

There are certain operational parameters that control the disk cache operation which are tunable. A number of *preset* parameter sets is provided, dependent on the size of the cache. These should suffice for most purposes, but under certain types of workload, it may be desirable to tune these parameters to better suite the particular workload patterns.

See **dcacheDevTune()** for description of the tunable parameters. It is recommended to call **dcacheShow()** after calling **dcacheTune()** in order to verify that the parameters where set as requested, and to inspect the cache statistics which may change dramatically.

Note that the hit ratio is a principal indicator of cache efficiency, and should be inspected during such tuning.

## BACKGROUND UPDATING

A dedicated task will be created to take care of updating the disk with blocks that have been modified in cache. The time period between updates is controlled with the tunable parameter *syncInterval*. Its priority should be set above the priority of any CPU-bound tasks so as to assure it can wake up frequently enough to keep the disk synchronized with the cache. There is only one such task for all cache devices configured. The task name is **tDcacheUpd**

The updating task also has the responsibility to invalidate disk cache blocks for removable devices which have not been used for 2 seconds or more.

There are a few global variables which control the parameters of this task, namely:

*dcacheUpdTaskPriority*

controls the default priority of the update task, and is set by default to 250.

*dcacheUpdTaskStack*

is used to set the update task stack size.

*dcacheUpdTaskOptions*

controls the task options for the update task.

All the above global parameters must be set prior to calling **dcacheDevCreate()** for the first time, with the exception of *dcacheUpdTaskPriority*, which may be modified in run-time, and takes effect almost immediately. It should be noted that this priority is not entirely fixed, at times when critical disk operations are performed, and **FIOFLUSH** ioctl is called, the caller task will temporarily *loan* its priority to the update task, to insure the completion of the flushing operation.

## REMOVABLE DEVICES

For removable devices, disk cache provides these additional features:

disk updating

is performed such that modified blocks will be written to disk within one second, so as to minimize the risk of losing data in case of a failure or disk removal.

error handling

includes a test for disk removal, so that if a disk is removed from the drive while an I/O operation is in progress, the disk removal event will be set immediately.

disk signature

which is a checksum of the disk's boot block, is maintained by the cache control structure, and it will be verified against the disk if it was idle for 2 seconds or more. Hence if during that idle time a disk was replaced, the change will be detected on the next disk access, and the condition will be flagged to the file system.

---

**NOTE:** It is very important that removable disks should all have a unique volume label, or volume serial number, which are stored in the disk's boot sector during formatting. Changing disks which have an identical boot sector may result in failure to detect the change, resulting in unpredictable behavior, possible file system corruption.

---

#### CACHE IMPLEMENTATION

Most Recently Used (MRU) disk blocks are stored in a collection of memory buffers called the disk cache. The purpose of the disk cache is to reduce the number of disk accesses and to accelerate disk read and write operations, by means of the following techniques:

- Most Recently Used blocks are stored in RAM, which results in the most frequently accessed data being retrieved from memory rather than from disk.
- Reading data from disk is performed in large units, relying on the read-ahead feature, one of the disk cache's tunable parameters.

Write operations are optimized because they occur to memory first. Then updating the disk happens in an orderly manner, by delayed write, another tunable parameter.

Overall, the main performance advantage arises from a dramatic reduction in the amount of time spent by the disk drive seeking, thus maximizing the time available for the disk to read and write actual data. In other words, you get efficient use of the disk drive's available throughput. The disk cache offers a number of operational parameters that can be tuned by the user to suit a particular file system workload pattern, for example, delayed write, read ahead, and bypass threshold.

The technique of delaying writes to disk means that if the system is turned off unexpectedly, updates that have not yet been written to the disk are lost. To minimize the effect of a possible crash, the disk cache periodically updates the disk. Modified blocks of data are not kept in memory more than a specified period of time. By specifying a small update period, the possible worst-case loss of data from a crash is the sum of changes possible during that specified period. For example, it is assumed that an update period of 2 seconds is sufficiently large to effectively optimize disk writes, yet small enough to make the potential loss of data a reasonably minor concern. It is possible to set the update period to 0, in which case, all updates are flushed to disk immediately. This is essentially the equivalent of using the **DOS\_OPT\_AUTOSYNC** option in earlier **dosFsLib** implementations. The disk cache allows you to negotiate between disk performance and memory consumption: The more memory allocated to the disk cache, the higher the "hit ratio" observed, which means increasingly better performance of file system operations. Another tunable parameter is the bypass threshold, which defines how much data constitutes a request large enough to justify bypassing the disk cache. When significantly large read or write requests are made by the application, the disk cache is circumvented and there is a direct transfer of data between the disk controller and the user data buffer. The use of bypassing, in conjunction with support for contiguous file allocation and access (via the **FIOCONTIG ioctl()** command and the **DOS\_O\_CONTIG open()** flag), should provide performance equivalent to that offered by the raw file system (rawFs).

**PARTITION INTERACTION**

The dcache CBIO layer is intended to operate atop an entire fixed disk device. When using the dcache layer with the dpart CBIO partition layer, it is important to place the dcache layer below the partition layer.

For example:

dosFsLib
dpart
dcache
blkloDev

**D****ENABLE/DISABLE THE DISK CACHE**

The function **dcacheDevEnable()** is used to enable the disk cache. The function **dcacheDevDisable()** is used to disable the disk cache. When the disk cache is disabled, all I/O will bypass the cache layer.

**SEE ALSO**      **dosFsLib, cbioLib, dpartCbio**

---

## dhcpcBootLib

**NAME**            **dhcpcBootLib** – DHCP boot-time client library

**ROUTINES**        **dhcpcBootInit()** - set up the DHCP client parameters and data structures  
**dhcpcBootBind()** - initialize the network with DHCP at boot time  
**dhcpcBootInformGet()** - obtain additional configuration parameters with DHCP

**DESCRIPTION**    This library contains the interface for the client side of the Dynamic Host Configuration Protocol (DHCP) used during system boot. DHCP is an extension of BOOTP, the bootstrap protocol. Like BOOTP, the protocol allows automatic system startup by providing an IP address, boot file name, and boot host's IP address over a network. Additionally, DHCP provides the complete set of configuration parameters defined in the Host Requirements RFCs and allows automatic reuse of network addresses by specifying a lease duration for a set of configuration parameters. This library is linked into the boot ROM image automatically if **INCLUDE\_DHCP** is defined at the time that image is constructed.

## HIGH-LEVEL INTERFACE

The VxWorks boot program uses this library to obtain configuration parameters with DHCP according to the client-server interaction detailed in RFC 2131 using the boot device specified in the boot parameters. The DHCP client supports devices attached to the IP protocol with the MUX/END interface. It also supports BSD Ethernet devices attached to the IP protocol.

To use DHCP, first build a boot ROM image with `INCLUDE_DHCPC` defined and set the appropriate flag in the boot parameters before initiating booting with the “@” command. The DHCP client will attempt to retrieve entries for the boot file name, and host IP address, as well as a subnet mask and broadcast address for the boot device. If a target IP address is not available, the client will retrieve those parameters in the context of a lease. Otherwise, it will search for permanent assignments using a simpler message exchange. Any entries retrieved with either method will only be used if the corresponding fields in the boot parameters are blank.

---

**NOTE:** After DHCP retrieves the boot parameters, the specified boot file is loaded and the system restarts. As a result, the boot-time DHCP client cannot renew any lease which may be associated with the assigned IP address. To avoid potential IP address conflicts while loading the boot file, the `DHCPC_MIN_LEASE` value should be set to exceed the file transfer time. In addition, the boot file must also contain the DHCP client library so that the lease obtained before the restart can be renewed. Otherwise, the network initialization using the boot parameters will fail. These restrictions do not apply if the target IP address is entered manually since the boot parameters do not involve a lease in that case.

---

**INCLUDE FILES**     `dhcpcBootLib.h`

**SEE ALSO**            `dhcpcLib`, RFC 1541, RFC 1533

---

# dhcpcCommonLib

**NAME**                `dhcpcCommonLib` – DHCP client interface shared code library

**ROUTINES**           `dhcpcOptionSet()` - add an option to the option request list  
`dhcpcOptionAdd()` - add an option to the client messages

**DESCRIPTION**        This library contains the shared functions used by the both the run-time and boot-time portions of the DHCP client.

**INCLUDE FILES**     `dhcpcLib.h`

**SEE ALSO**            `dhcpcLib`

---

## dhcpcLib

<b>NAME</b>	<b>dhcpcLib</b> – Dynamic Host Configuration Protocol (DHCP) run-time client API
<b>ROUTINES</b>	<b>dhcpcLibInit()</b> - DHCP client library initialization <b>dhcpcInit()</b> - assign network interface and setup lease request <b>dhcpcEventHookAdd()</b> - add a routine to handle configuration parameters <b>dhcpcEventHookDelete()</b> - remove the configuration parameters handler <b>dhcpcCacheHookAdd()</b> - add a routine to store and retrieve lease data <b>dhcpcCacheHookDelete()</b> - delete a lease data storage routine <b>dhcpcBind()</b> - obtain a set of network configuration parameters with DHCP <b>dhcpcVerify()</b> - renew an established lease <b>dhcpcRelease()</b> - relinquish specified lease <b>dhcpcInformGet()</b> - obtain additional configuration parameters with DHCP <b>dhcpcShutdown()</b> - disable DHCP client library <b>dhcpcOptionGet()</b> - retrieve an option provided to a client and store in a buffer <b>dhcpcServerGet()</b> - retrieve the current DHCP server <b>dhcpcTimerGet()</b> - retrieve current lease timers <b>dhcpcParamsGet()</b> - retrieve current configuration parameters

**DESCRIPTION**

This library implements the run-time access to the client side of the Dynamic Host Configuration Protocol (DHCP). DHCP is an extension of BOOTP. Like BOOTP, the protocol allows a host to initialize automatically by obtaining its IP address, boot file name, and boot host's IP address over a network. Additionally, DHCP provides a client with the complete set of parameters defined in the Host Requirements RFCs and allows automatic reuse of network addresses by specifying individual leases for each set of configuration parameters. The compatible message format allows DHCP participants to interact with BOOTP participants. The **dhcpcLibInit()** routine links this library into the VxWorks image. This happens automatically if **INCLUDE\_DHCP** is defined at the time the image is built.

### CONFIGURATION INTERFACE

When used during run time, the DHCP client library establishes and maintains one or more DHCP leases. Each lease provides access to a set of configuration parameters. If requested, the parameters retrieved will be used to reconfigure the associated network interface, but may also be handled separately through an event hook. The **dhcpcEventHookAdd()** routine specifies a function which is invoked whenever the lease status changes. The **dhcpcEventHookDelete()** routine will disable that notification. The automatic reconfiguration must be limited to one lease for a particular network interface. Otherwise, multiple leases would attempt to reconfigure the same device, with unpredictable results.

#### HIGH-LEVEL INTERFACE

To access the DHCP client during run time, an application must first call the **dhcpcInit()** routine with a pointer to the network interface to be used for communication with a DHCP server. Each call to the initialization routine returns a unique identifier to be used in subsequent calls to the DHCP client routines. Next, the application must specify a client identifier for the lease using the **dhcpcOptionSet()** call. Typically, the link-level hardware address is used for this purpose. Additional calls to the option set routine may be used to request specific DHCP options. After all calls to that routine are completed, a call to **dhcpcBind()** will retrieve a set of configuration parameters according to the client-server interaction detailed in RFC 1541.

Each sequence of the three function calls described above, if successful, will retrieve a set of configuration parameters from a DHCP server. The **dhcpcServerGet()** routine retrieves the address of the server that provided a particular lease. The **dhcpcTimerGet()** routine will retrieve the current values for both lease timers.

Alternatively, the **dhcpcParamsGet()** and **dhcpcOptionGet()** routines will access any options provided by a DHCP server. In addition to the lease identifier obtained from the initialization routine, the **dhcpcParamsGet()** routine accepts a parameter descriptor structure that selects any combination of the options described in RFC 1533 for retrieval. Similarly, the **dhcpcOptionGet()** routine retrieves the values associated with a single option.

#### LOW-LEVEL INTERFACE

This library also contains several routines which explicitly generate DHCP messages. The **dhcpcVerify()** routine causes the client to renew a particular lease, regardless of the time remaining. The **dhcpcRelease()** routine relinquishes the specified lease. The associated parameters are no longer valid. If those parameters were used by the underlying network device, the routine also shuts off all network processing for that interface. Finally, the **dhcpcShutdown()** routine will release all active leases and disable all the DHCP client library routines.

#### OPTIONAL INTERFACE

The **dhcpcCacheHookAdd()** routine registers a function that the client will use to store and retrieve lease data. The client can then re-use this information if it is rebooted. The **dhcpcCacheHookDelete()** routine prevents the re-use of lease data. Initially, a function to access permanent storage is not provided.

**INCLUDE FILES**     **dhcpcLib.h**

**SEE ALSO**         RFC 1541, RFC 1533

---

## dhcpcShow

<b>NAME</b>	<b>dhcpcShow</b> – DHCP run-time client information display routines
<b>ROUTINES</b>	<b>dhcpcShowInit()</b> - initialize the DHCP show facility <b>dhcpcServerShow()</b> - display current DHCP server <b>dhcpcTimersShow()</b> - display current lease timers <b>dhcpcParamsShow()</b> - display current lease parameters
<b>DESCRIPTION</b>	This library provides routines that display various data related to the DHCP run-time client library such as the lease timers and responding server. The <b>dhcpcShowInit()</b> routine links the show facility into the VxWorks image. This happens automatically if <b>INCLUDE_NET_SHOW</b> and <b>INCLUDE_DHCP</b> are defined at the time the image is built.
<b>INCLUDE FILES</b>	<b>dhcpcLib.h</b>
<b>SEE ALSO</b>	<b>dhcpcLib</b>

---

## dhcprLib

<b>NAME</b>	<b>dhcprLib</b> – DHCP relay agent library
<b>ROUTINES</b>	No Callable Routines
<b>DESCRIPTION</b>	This library implements a relay agent for the Dynamic Host Configuration Protocol (DHCP). DHCP is an extension of BOOTP. Like BOOTP, it allows a target to configure itself dynamically by using the network to get its IP address, a boot file name, and the DHCP server's address. The relay agent forwards DHCP messages between clients and servers resident on different subnets. The standard DHCP server, if present on a subnet, can also forward messages across subnet boundaries. The relay agent is needed only if there is no DHCP server running on the subnet. The <b>dhcprLibInit()</b> routine links this library into the VxWorks system. This happens automatically if <b>INCLUDE_DHCP</b> is defined at the time the system is built, as long as <b>INCLUDE_DHCP</b> is <i>not</i> also defined.
<b>HIGH-LEVEL INTERFACE</b>	The <b>dhcprInit()</b> routine initializes the relay agent automatically. The relay agent forwards incoming DHCP messages to the IP addresses specified at build time in <b>dhcprTargetTbl[]</b> .
<b>INCLUDE FILES</b>	<b>dhcprLib.h</b>
<b>SEE ALSO</b>	RFC 1541, RFC 1533

---

## dhcpsLib

**NAME**            **dhcpsLib** – Dynamic Host Configuration Protocol (DHCP) server library

**ROUTINES**        **dhcpsInit()** - set up the DHCP server parameters and data structures  
**dhcpsLeaseEntryAdd()** - add another entry to the address pool  
**dhcpsLeaseHookAdd()** - assign a permanent lease storage hook for the server  
**dhcpsAddressHookAdd()** - assign a permanent address storage hook for the server

**DESCRIPTION**    This library implements the server side of the Dynamic Host Configuration Protocol (DHCP). DHCP is an extension of BOOTP. Like BOOTP, it allows a target to configure itself dynamically by using the network to get its IP address, a boot file name, and the DHCP server's address. Additionally, DHCP provides for automatic reuse of network addresses by specifying individual leases as well as many additional options. The compatible message format allows DHCP participants to inter-operate with BOOTP participants. The **dhcpsInit()** routine links this library into the VxWorks image. This happens automatically if **INCLUDE\_DHCP** is defined when the image is built.

### PRIMARY INTERFACE

The **dhcpsInit()** routine initializes the server. It reads the hard-coded server configuration data that is stored in three separate tables. The first table contains entries as follows:

```
DHCPS_LEASE_DESC dhcpsLeaseTbl [] =
{
  {"sample1", "90.11.42.24", "90.11.42.24", "clid=\1:0x08003D21FE90\""},
  {"sample2", "90.11.42.25", "90.11.42.28", "maxl=90:dfl=60"},
  {"sample3", "90.11.42.29", "90.11.42.34", "maxl=0xffffffff:file=/vxWorks"},
  {"sample4", "90.11.42.24", "90.11.42.24", "albp=true:file=/vxWorks"}
};
```

Each entry contains a name of up to eight characters, the starting and ending IP addresses of a range, and the parameters associated with the lease. The four samples shown demonstrate the four types of leases.

Manual leases contain a specific client ID, and are issued only to that client, with an infinite duration. The example shown specifies a MAC address, which is the identifier type used by the VxWorks DHCP client.

Dynamic leases specify a finite maximum length, and can be issued to any requesting client. These leases allow later re-use of the assigned IP address. If not explicitly specified in the parameters field, these leases use the values of **DHCPS\_MAX\_LEASE** and **DHCPS\_DFLT\_LEASE** to determine the lease length.

Automatic leases are implied by the infinite maximum length. Their IP addresses are assigned permanently to any requesting client.

The last sample demonstrates a lease that is also available to BOOTP clients. The infinite maximum length is implied, and any timing-related parameters are ignored.

The DHCP server supplies leases to DHCP clients according to the lease type in the order shown above. Manual leases have the highest priority and leases available to BOOTP clients the lowest.

Entries in the parameters field may be one of these types:

**bool**

Takes values of “true” or “false”, for example, ipfd=true. Unrecognized values default to false.

**str**

Takes a character string as a value, for example, hstn=“clapton”. If the string includes a delimiter character, such as a colon, it should be enclosed in quotation marks.

**octet**

Takes an 8-bit integer in decimal, octal, or hexadecimal, for example, 8, 070, 0xff.

**short**

Takes a 16-bit integer.

**long**

Takes a 32-bit integer.

**ip**

Takes a string that is interpreted as a 32-bit IP address. One of the following formats is expected: a.b.c.d, **a.b.c** or a.b. In the second format, c is interpreted as a 16-bit value. In the third format, b is interpreted as a 24-bit value, for example siad=90.11.42.1.

**iplist**

Takes a list of IP addresses, separated by white space, for example, rout=133.4.31.1 133.4.31.2 133.4.31.3.

**ippairs**

Takes a list of IP address pairs. Each IP address is separated by white space and grouped in pairs, for example, strt=133.4.27.0 133.4.31.1 133.4.36.0 133.4.31.1.

**mtpt**

Takes a list of 16 bit integers, separated by white space, for example, mtpt=1 2 3 4 6 8.

**clid**

Takes a client identifier as a value. Client identifiers are represented by the quoted string “*type:data*”, where *type* is an integer from 0 to 255, as defined by the IANA, and *data* is a sequence of 8-bit values in hexadecimal. The client ID is usually a MAC address, for example, clid=“1:0x08004600e5d5”.

The following table lists the option specifiers and descriptions for every possible entry in the parameter list. When available, the option code from RFC 2132 is included.

<b>Name</b>	<b>Code</b>	<b>Type</b>	<b>Description</b>
sname	-	str	Optional server name.
file	-	str	Name of file containing the boot image.
siad	-	ip	Address of server that offers the boot image.
albp	-	bool	If true, this entry is also available to BOOTP clients. For entries using static allocation, this value becomes true by default and <i>maxl</i> becomes infinity.
maxl	-	long	Maximum lease duration in seconds.
dfl	-	long	Default lease duration in seconds. If a client does not request a specific lease duration, the server uses this value.
clid	-	clid	This specifies a client identifier for manual leases. The VxWorks client uses a MAC address as the client identifier.
pmid	-	clid	This specifies a client identifier for client-specific parameters to be included in a lease. It should be present in separate entries without IP addresses.
clas	-	str	This specifies a class identifier for class-specific parameters to be included in a lease. It should be present in separate entries without IP addresses.
snmk	1	ip	Subnet mask of the IP address to be allocated. The default is a natural mask corresponding to the IP address. The server will not issue IP addresses to clients on different subnets.
tmof	2	long	Time offset from UTC in seconds.
rout	3	iplist	A list of routers on the same subnet as the client.
tmsv	4	iplist	A list of time servers (RFC 868).
nmsv	5	iplist	A list of name servers (IEN 116).
dnsv	6	iplist	A list of DNS servers (RFC 1035).
lgsv	7	iplist	A list of MIT-LCS UDP log servers.
cksv	8	iplist	A list of Cookie servers (RFC 865).
lpsv	9	iplist	A list of LPR servers (RFC 1179).
imsv	10	iplist	A list of Imagen Impress servers.
rlsv	11	iplist	A list of Resource Location servers (RFC 887).
hstn	12	str	Hostname of the client.
btsz	13	short	Size of boot image.
mdmp	14	str	Path name to which client dumps core.
dnsd	15	str	Domain name for DNS.
swsv	16	ip	IP address of swap server.
rpth	17	str	Path name of root disk of the client.
epth	18	str	Extensions Path (See RFC 1533).
ipfd	19	bool	If true, the client performs IP forwarding.
nlsr	20	bool	If true, the client can perform non-local source routing.
ply	21	ippairs	Policy filter for non-local source routing. A list of pairs of (Destination IP, Subnet mask).

Name	Code	Type	Description
mdgs	22	short	Maximum size of IP datagram that the client should be able to reassemble.
ditl	23	octet	Default IP TTL.
mtat	24	long	Aging timeout (in seconds) to be used with Path MTU discovery (RFC 1191).
mtpt	25	mtpt	A table of MTU sizes to be used with Path MTU Discovery.
ifmt	26	short	MTU to be used on an interface.
asnl	27	bool	If true, the client assumes that all subnets to which the client is connected use the same MTU.
brda	28	ip	Broadcast address in use on the client's subnet. The default is calculated from the subnet mask and the IP address.
mskd	29	bool	If true, the client should perform subnet mask discovery using ICMP.
msks	30	bool	If true, the client should respond to subnet mask requests using ICMP.
rtrd	31	bool	If true, the client should solicit routers using Router Discovery defined in RFC 1256.
rtsl	32	ip	Destination IP address to which the client sends router solicitation requests.
strt	33	ippairs	A table of static routes for the client, which are pairs of (Destination, Router). It is illegal to specify default route as a destination.
trlr	34	bool	If true, the client should negotiate the use of trailers with ARP (RFC 893).
arpt	35	long	Timeout in seconds for ARP cache.
encp	36	bool	If false, the client uses RFC 894 encapsulation. If true, it uses RFC 1042 (IEEE 802.3) encapsulation.
dttl	37	octet	Default TTL of TCP.
kain	38	long	Interval of the client's TCP keepalive in seconds.
kagb	39	bool	If true, the client should send TCP keepalive messages with a octet of garbage for compatibility.
nisd	40	str	Domain name for NIS.
nisv	41	iplist	A list of NIS servers.
ntsv	42	iplist	A list of NTP servers.
nnsv	44	iplist	A list of NetBIOS name server. (RFC 1001, 1002)
ndsv	45	iplist	A list of NetBIOS datagram distribution servers (RFC 1001, 1002).
nbnt	46	octet	NetBIOS node type (RFC 1001, 1002).
nbsc	47	str	NetBIOS scope (RFC 1001, 1002).
xfsv	48	iplist	A list of font servers of X Window system.
xdmn	49	iplist	A list of display managers of X Window system.

Name	Code	Type	Description
dht1	58	short	This value specifies when the client should start RENEWING. The default of 500 means the client starts RENEWING after 50% of the lease duration passes.
dht2	59	short	This value specifies when the client should start REBINDING. The default of 875 means the client starts REBINDING after 87.5% of the lease duration passes.

Finally, to function correctly, the DHCP server requires access to some form of permanent storage. The `DHCPS_LEASE_HOOK` constant specifies the name of a storage routine with the following interface:

```
STATUS dhcpsStorageHook (int op, char *buffer, int datalen);
```

The storage routine is installed by a call to the `dhcpsLeaseHookAdd()` routine. The manual pages for `dhcpsLeaseHookAdd()` describe the parameters and required operation of the storage routine.

#### SECONDARY INTERFACE

In addition to the hard-coded entries, address entries may be added after the server has started by calling the following routine:

```
STATUS dhcpsLeaseEntryAdd (char *name, char *start, char *end, char *config);
```

The parameters specify an entry name, starting and ending values for a block of IP addresses, and additional configuration information in the same format as shown above for the hard-coded entries. Each parameter must be formatted as a NULL-terminated string.

The `DHCPS_ADDRESS_HOOK` constant specifies the name of a storage routine, used to preserve address entries added after startup, which has the following prototype:

```
STATUS dhcpsAddressStorageHook (int op,  
                                char *name, char *start, char *end,  
                                char *params);
```

The storage routine is installed with the `dhcpsAddressHookAdd()` routine, and is fully described in the manual pages for that function.

#### OPTIONAL INTERFACE

The DHCP server can also receive messages forwarded from different subnets by a relay agent. To provide addresses to clients on different subnets, the appropriate relay agents must be listed in the provided table in `usrNetwork.c`. A sample configuration is:

```
DHCPS_RELAY_DESC dhcpsRelayTbl [] =  
{  
  {"90.11.46.75", "90.11.46.0"}  
};
```

Each entry in the table specifies the address of a relay agent that will transmit the request and the corresponding subnet number. To issue leases successfully, the address pool must also contain IP addresses for the monitored subnets.

The following table allows a DHCP server to act as a relay agent in addition to its default function of processing messages. It consists of a list of IP addresses.

```
DHCP_TARGET_DESC dhcpTargetTbl [] =
{
    {"90.11.43.2"},
    {"90.11.44.1"}
};
```

Each IP address in this list receives a copy of any client messages generated on the subnets monitored by the server.

**INCLUDE FILES**    **dhcpsLib.h**

**SEE ALSO**        RFC 1541, RFC 1533

---

## dirLib

**NAME**            **dirLib** – directory handling library (POSIX)

**ROUTINES**        **opendir()** - open a directory for searching (POSIX)  
**readdir()** - read one entry from a directory (POSIX)  
**rewinddir()** - reset position to the start of a directory (POSIX)  
**closedir()** - close a directory (POSIX)  
**fstat()** - get file status information (POSIX)  
**stat()** - get file status information using a pathname (POSIX)  
**fstatfs()** - get file status information (POSIX)  
**statfs()** - get file status information using a pathname (POSIX)  
**utime()** - update time on a file

**DESCRIPTION**    This library provides POSIX-defined routines for opening, reading, and closing directories on a file system. It also provides routines to obtain more detailed information on a file or directory.

**SEARCHING DIRECTORIES**  
Basic directory operations, including **opendir()**, **readdir()**, **rewinddir()**, and **closedir()**, determine the names of files and subdirectories in a directory.

A directory is opened for reading using **opendir()**, specifying the name of the directory to be opened. The **opendir()** call returns a pointer to a directory descriptor, which identifies a directory stream. The stream is initially positioned at the first entry in the directory.

Once a directory stream is opened, **readdir()** is used to obtain individual entries from it. Each call to **readdir()** returns one directory entry, in sequence from the start of the directory. The **readdir()** routine returns a pointer to a **dirent** structure, which contains the name of the file (or subdirectory) in the **d\_name** field.

The **rewinddir()** routine resets the directory stream to the start of the directory. After **rewinddir()** has been called, the next **readdir()** will cause the current directory state to be read in, just as if a new **opendir()** had occurred. The first entry in the directory will be returned by the first **readdir()**.

The directory stream is closed by calling **closedir()**.

#### GETTING FILE INFORMATION

The directory stream operations described above provide a mechanism to determine the names of the entries in a directory, but they do not provide any other information about those entries. More detailed information is provided by **stat()** and **fstat()**.

The **stat()** and **fstat()** routines are essentially the same, except for how the file is specified. The **stat()** routine takes the name of the file as an input parameter, while **fstat()** takes a file descriptor number as returned by **open()** or **creat()**. Both routines place the information from a directory entry in a **stat** structure whose address is passed as an input parameter. This structure is defined in the include file **stat.h**. The fields in the structure include the file size, modification date/time, whether it is a directory or regular file, and various other values.

The **st\_mode** field contains the file type; several macro functions are provided to test the type easily. These macros operate on the **st\_mode** field and evaluate to **TRUE** or **FALSE** depending on whether the file is a specific type. The macro names are:

#### **S\_ISREG**

test if the file is a regular file

#### **S\_ISDIR**

test if the file is a directory

#### **S\_ISCHR**

test if the file is a character special file

#### **S\_ISBLK**

test if the file is a block special file

#### **S\_ISFIFO**

test if the file is a FIFO special file

Only the regular file and directory types are used for VxWorks local file systems. However, the other file types may appear when getting file status from a remote file system (using NFS).

As an example, the `S_ISDIR` macro tests whether a particular entry describes a directory. It is used as follows:

```
char          *filename;
struct stat   fileStat;
stat (filename, &fileStat);
if (S_ISDIR (fileStat.st_mode))
    printf ("%s is a directory.\n", filename);
else
    printf ("%s is not a directory.\n", filename);
```

See the `ls()` routine in `usrLib` for an illustration of how to combine the directory stream operations with the `stat()` routine.

INCLUDE FILES `dirent.h, stat.h`

---

## distIfShow

**NAME** `distIfShow` – distributed objects interface adapter show routines (VxFusion Opt.)

**ROUTINES** `distIfShow()` - display information about the installed interface adapter (VxFusion Opt.)

**DESCRIPTION** This library provides a show routine for displaying information about the installed interface adapter.

**AVAILABILITY** This module is distributed as a component of the unbundled distributed message queues option, VxFusion.

**INCLUDE FILES** `distIfLib.h`

**SEE ALSO** `distStatLib`

---

## distLib

**NAME** `distLib` – distributed objects initialization and control library (VxFusion Opt.)

**ROUTINES** `distInit()` - initialize and bootstrap the current node (VxFusion Opt.)  
`distCtl()` - perform a distributed objects control function (VxFusion Opt.)

**DESCRIPTION** This library provides an initialization and control interface for VxFusion.

Use **distInit()** to initialize VxFusion on the current node. In addition to performing local initialization, **distInit()** attempts to locate remote VxFusion nodes on the network and download copies of the databases from one of the remote nodes.

Call **distCtl()** to set VxFusion run-time parameters using an **ioctl()**-like syntax.

---

**NOTE:** In this release, the **distInit()** routine is called automatically with default parameters when a target boots using a VxWorks image with VxFusion installed.

---

<b>AVAILABILITY</b>	This module is distributed as a component of the unbundled distributed message queues option, VxFusion.
<b>INCLUDE FILES</b>	<b>distLib.h</b>

---

## distNameLib

<b>NAME</b>	<b>distNameLib</b> – distributed name database library (VxFusion Opt.)
<b>ROUTINES</b>	<b>distNameAdd()</b> - add an entry to the distributed name database (VxFusion Opt.) <b>distNameFind()</b> - find an object by name in the local database (VxFusion Opt.) <b>distNameFindByValueAndType()</b> - look up the name of an object by value and type (VxFusion Opt.) <b>distNameRemove()</b> - remove an entry from the distributed name database (VxFusion Opt.)
<b>DESCRIPTION</b>	This library contains the distributed objects distributed name database and routines for manipulating it. Symbolic names are bound to values, such as message queue identifiers or simple integers. Entries can be found by name or by value and type. The distributed name database is replicated throughout the system, with a copy sitting on each node.  The distributed name database library is initialized by calling <b>distInit()</b> in <b>distLib</b> .
<b>AVAILABILITY</b>	This module is distributed as a component of the unbundled distributed message queues option, VxFusion.
<b>INCLUDE FILES</b>	<b>distNameLib.h</b>
<b>SEE ALSO</b>	<b>distLib</b> , <b>distNameShow</b>

---

## distNameShow

<b>NAME</b>	<b>distNameShow</b> – distributed name database show routines (VxFusion Opt.)
<b>ROUTINES</b>	<b>distNameShow()</b> - display the entire distributed name database (VxFusion Opt.) <b>distNameFilterShow()</b> - display the distributed name database filtered by type (VxFusion Opt.)
<b>DESCRIPTION</b>	This library provides routines for displaying the contents of the distributed name database.
<b>AVAILABILITY</b>	This module is distributed as a component of the unbundled distributed message queues option, VxFusion.
<b>INCLUDE FILES</b>	<b>distNameLib.h</b>
<b>SEE ALSO</b>	<b>distNameLib</b>

---

## distTBufLib

<b>NAME</b>	<b>distTBufLib</b> – distributed objects telegram buffer library (VxFusion Opt.)
<b>ROUTINES</b>	<b>distTBufAlloc()</b> - allocate a telegram buffer from the pool of buffers (VxFusion Opt.) <b>distTBufFree()</b> - return a telegram buffer to the pool of buffers (VxFusion Opt.)
<b>DESCRIPTION</b>	This library provides routines for allocating and freeing telegram buffers. Telegrams are the largest packets that can be sent between nodes by the distributed objects product; their size is limited by the MTU size of the underlying communications. If a distributed objects message exceeds the space allocated in a telegram for message data, that message is divided into multiple telegrams that are sent out in sequence.
<b>AVAILABILITY</b>	This module is distributed as a component of the unbundled distributed message queues option, VxFusion.
<b>INCLUDE FILES</b>	<b>distTBufLib.h</b>

## dosFsFmtLib

<b>NAME</b>	<b>dosFsFmtLib</b> – MS-DOS media-compatible file system formatting library
<b>ROUTINES</b>	<b>dosFsVolFormat()</b> - format an MS-DOS compatible volume
<b>DESCRIPTION</b>	<p>This module is a scalable companion module for <b>dosFsLib</b>, and is intended to facilitate high level formatting of disk volumes.</p> <p>There are two ways to high level format a volume:</p> <ol style="list-style-type: none"><li>(1) Directly calling <b>dosFsVolFormat()</b> routine allows to have complete control over the format used, parameters and allows to supply a hook routine which for instance could interactively prompt the user to modify disk parameters.</li><li>(2) Calling ioctl command <b>FIODISKINIT</b> will invoke the formatting routine via <b>dosFsLib</b>. This uses the default volume format and parameters.</li></ol>
<b>AVAILABILITY</b>	<p>This routine is an optional part of the MS-DOS file system, and may be included in a target system if it is required to be able to format new volumes.</p> <p>In order to include this option, the following function needs to be invoked during system initialization:</p> <pre>void dosFsFmtLibInit( void );</pre> <p>See reference page <b>dosFsVolFormat()</b> for complete description of supported formats, options and arguments.</p>
<b>SEE ALSO</b>	<b>dosFsLib</b>

---

## dosFsLib

<b>NAME</b>	<b>dosFsLib</b> – MS-DOS media-compatible file system library
<b>ROUTINES</b>	<b>dosSetVolCaseSens()</b> - set case sensitivity of volume <b>dosFsVolDescGet()</b> - convert a device name into a DOS volume descriptor pointer. <b>dosFsChkDsk()</b> - make volume integrity checking. <b>dosFsLastAccessDateEnable()</b> - enable last access date updating for this volume <b>dosFsLibInit()</b> - prepare to use the dosFs library <b>dosFsDevCreate()</b> - create file system device. <b>dosFsShow()</b> - display dosFs volume configuration data.

**DESCRIPTION** This library implements the MS-DOS compatible file system. This is a multi-module library, which depends on sub-modules to perform certain parts of the file system functionality. A number of different file system format variations are supported.

#### USING THIS LIBRARY

The various routines provided by the VxWorks DOS file system (dosFs) may be separated into three broad groups: general initialization, device initialization, and file system operation.

The **dosFsLibInit()** routine is the principal initialization function; it should be called once during system initialization, regardless of how many dosFs devices are to be used.

Another dosFs routine is used for device initialization. For each dosFs device, **dosFsDevCreate()** must be called to install the device in VxWorks device list. In the case where partitioned disks are used, **dosFsDevCreate()** must be called for each partition that is anticipated, thereby it is associated with a logical device name, so it can be later accessed via the I/O system.

In case of a removable disk, **dosFsDevCreate()** must be called during system initialization time, even if a cartridge or diskette may be absent from the drive at boot time. **dosFsDevCreate()** will only associate the device with a logical device name. Device access will be done only when the logical device is first accessed by the application.

More detailed information on all of these routines is provided below.

#### INITIALIZING DOSFSLIB

To enable this file system in a particular VxWorks configuration, a library initialization routine must be called for each sub-module of the file system, as well as for the underlying disk cache, partition manager and drivers. This is usually done at system initialization time, within the *usrRoot* task context.

Following is the list of initialization routines that need to be called:

##### **dosFsLibInit()**

(mandatory) initialize the principle dosFs module. Must be called first.

##### **dosFsFatInit()**

(mandatory) initialize the File Allocation Table handler, which supports 12-bit, 16-bit and 32-bit FATs.

##### **dosVDirLibInit()**

(choice) install the variable size directory handler supporting Windows-compatible Long File Names (VFAT) Directory Handler.

##### **dosDirOldLibInit()**

(choice) install the fixed size directory handler which supports old-fashioned 8.3 MS-DOS file names, and Wind River Systems proprietary long file names (VXLONG).

##### **dosFsFmtLibInit()**

(optional) install the volume formatting module.

**dosFsLib****dosChkLibInit()**

(optional) install the file system consistency checking module.

The two Directory handlers which are marked *choice* are installed in accordance with the system requirements, either one of these modules could be installed or both, in which case the VFAT will take precedence for MS-DOS compatible volumes.

Also, at least one *CBIO* module must be initialized on a per-device basis prior to calling **dosFsDevCreate()**. See the related documentation for more details and examples.

**DEFINING A DOSFS DEVICE**

The **dosFsDevCreate()** routine associates a device with the **dosFsLib** functions. It expects three parameters:

- (1) A pointer to a name string, to be used to identify the device - logical device name. This will be part of the pathname for I/O operations which operate on the device. This name will appear in the I/O system device table, which may be displayed using the **iosDevShow()** routine.
- (2) **CBIO\_DEV\_ID** - a pointer to the **CBIO\_DEV** structure which provides interface to particular disk, via a disk cache, or a partition manager or a combination of a number of *CBIO* modules which are stacked on top of each other to form one of many configurations possible.
- (3) A maximum number of files can be simultaneously opened on a particular device.
- (4) Because volume integrity check utility can be automatically invoked every time a device is mounted, this parameter indicates whether the consistency check needs to be performed automatically on a given device, and on what level of verbosity is required. In any event, the consistency check may be invoked at a later time, *e.g.*, by calling **chkdsk()**. See description for **FIOCHKDSK** ioctl command for more information.

For example:

```
dosFsDevCreate
(
    "/sd0",          /* name to be used for volume */
    pCbio,          /* pointer to device descriptor */
    10,             /* max no. of simultaneously open files */
    DOS_CHK_REPAIR | DOS_CHK_VERB_1
                    /* check volume during mounting and repair */
                    /* errors, and display volume statistics */
)
```

Once **dosFsDevCreate()** has been called, the device can be accessed using *ioLib* generic I/O routines: **open()**, **read()**, **write()**, **close()**, **ioctl()**, **remove()**. Also, the user-level utility functions may be used to access the device at a higher level (See **usrFsLib** reference page for more details).

## DEVICE AND PATH NAMES

On true MS-DOS machines, disk device names are typically of the form “A:”, that is, a single letter designator followed by a colon. Such names may be used with the VxWorks dosFs file system. However, it is possible (and desirable) to use longer, more mnemonic device names, such as **DOS1:**, or **/floppy0**. The name is specified during the **dosFsDevCreate()** call.

The pathnames used to specify dosFs files and directories may use either forward slashes (“/”) or backslashes (“\”) freely mixed. The choice of forward slashes or backslashes has absolutely no effect on the directory data written to the disk. (Note, however, that forward slashes are not allowed within VxWorks dosFs filenames, although they are normally legal for pure MS-DOS implementations.)

For the sake of consistency however use of forward slashes (“/”) is recommended at all times.

The leading slash of a dosFs pathname following the device name is optional. For example, both **DOS1:newfile.new** and **DOS1:/newfile.new** refer to the same file.

## USING EXTENDED DIRECTORY STRUCTURE

This library supports DOS4.0 standard file names which fit the restrictions of eight upper-case characters optionally followed by a three-character extension, as well as Windows style VFAT standard long file names that are stored mixed cased on disk, but are case insensitive when searched and matched (*e.g.*, during **open()** call). The VFAT long file name is stored in a variable number of consecutive directory entries. Both standards restrict file size to 4 GB (32 bit value).

To provide additional flexibility, this implementation of the DOS file system provides proprietary long file name format (VXLONGNAMES), which uses a simpler directory structure: the directory entry is of fixed size. When this option is used, file names may consist of any sequence of up to 40 ASCII characters. No case conversion is performed, and file name match is case-sensitive. With this directory format the file maximum size is expanded to 1 Terabyte (40 bit value).

---

**NOTE:** Because special directory entries are used on the disk, disks which use the extended names are *not* compatible with other implementation of the MS-DOS systems, and cannot be read on MS-DOS or Windows machines.

---

To enable the extended file names, set the **DOS\_OPT\_VXLONGNAMES** flag when calling **dosFsVolFormat()**.

## READING DIRECTORY ENTRIES

Directories on VxWorks dosFs volumes may be searched using the **opendir()**, **readdir()**, **rewinddir()**, and **closedir()** routines. These calls allow the names of files and subdirectories to be determined.

To obtain more detailed information about a specific file, use the **fstat()** or **stat()** routine. Along with standard file information, the structure used by these routines also returns the file attribute byte from a dosFs directory entry.

For more information, see the manual entry for **dirLib**.

#### FILE DATE AND TIME

Directory entries on dosFs volumes contain creation, last modification time and date, and the last access date for each file or subdirectory. Directory last modification time and date fields are set only when a new entry is created, but not when any directory entries are deleted. The last access date field indicates the date of the last read or write. The last access date field is an optional field, per Microsoft. By default, file open-read-close operations do not update the last access date field. This default avoids media writes (writing out the date field) during read only operations. In order to enable the updating of the optional last access date field for open-read-close operations, you must call **dosFsLastAccessDateEnable()**, passing it the volumes **DOS\_VOLUME\_DESC\_ID** and **TRUE**.

The dosFs file system uses the ANSI **time()** function, that returns system clock value to obtain date and time. It is recommended that the target system should set the system time during system initialization time from a network server or from an embedded Calendar / Clock hardware component, so that all files on the file system would be associated with a correct date and time.

The file system consistency checker (see below) sets system clock to value following the latest date-time field stored on the disk, if it discovers, that function **time()** returns a date earlier than Jan 1, 1998, meaning that the target system does not have a source of valid date and time to synchronize with.

See also the reference manual entry for **ansiTime**.

#### FILE ATTRIBUTES

Directory entries on dosFs volumes contain an attribute byte consisting of bit-flags which specify various characteristics of the entry. The attributes which are identified are: read-only file, hidden file, system file, volume label, directory, and archive. The VxWorks symbols for these attribute bit-flags are:

##### **DOS\_ATTR\_RDONLY**

File is write-protected, can not be modified or deleted.

##### **DOS\_ATTR\_HIDDEN**

this attribute is not used by VxWorks.

##### **DOS\_ATTR\_SYSTEM**

this attribute is not used by VxWorks.

##### **DOS\_ATTR\_VOL\_LABEL**

directory entry describes a volume label, this attribute can not be set or used directly, see **ioctl()** command **FIOLABELGET** and **FIOLABELSET** below for volume label manipulation.

**DOS\_ATTR\_DIRECTORY**

directory entry is a subdirectory, this attribute can not be set directly.

**DOS\_ATTR\_ARCHIVE**

this attribute is not used by VxWorks.

All the flags in the attribute byte, except the directory and volume label flags, may be set or cleared using the `ioctl()` `FIOATTRIBSET` function. This function is called after opening the specific file whose attributes are to be changed. The attribute byte value specified in the `FIOATTRIBSET` call is copied directly. To preserve existing flag settings, the current attributes should first be determined via `fstat()`, and the appropriate flag(s) changed using bitwise AND or OR operations. For example, to make a file read-only, while leaving other attributes intact:

```

struct stat fileStat;
fd = open ("file", O_RDONLY, 0);      /* open file          */
fstat (fd, &fileStat);                /* get file status    */
ioctl (fd, FIOATTRIBSET, (fileStat.st_attr | DOS_ATTR_RDONLY));
                                     /* set read-only flag */
close (fd);                            /* close file         */

```

See also the reference manual entry for `attrib()` and `xattrib()` for user-level utility routines which control the attributes of files or file hierarchy.

**CONTIGUOUS FILE SUPPORT**

The VxWorks dosFs file system provides efficient files storage: space will be allocated in groups of clusters (also termed *extents*) so that a file will be composed of relatively large contiguous units. This nearly contiguous allocation technique is designed to effectively eliminate the effects of disk space fragmentation, keeping throughput very close to the maximum of which the hardware is capable of.

However dosFs provides mechanism to allocate truly contiguous files, meaning files which are made up of a consecutive series of disk sectors. This support includes both the ability to allocate contiguous space to a file and optimized access to such a file when it is used. Usually this will somewhat improve performance when compared to Nearly Contiguous allocation, at the price of disk space fragmentation.

To allocate a contiguous area to a file, the file is first created in the normal fashion, using `open()` or `creat()`. The file descriptor returned during the creation of the file is then used to make an `ioctl()` call, specifying the `FIOCONTIG` or `FIOCONTIG64` function. The last parameter to the `FIOCONTIG` function is the size of the requested contiguous area in bytes. If the `FIOCONTIG64` is used, the last parameter is pointer to 64-bit integer variable, which contains the required file size. It is also possible to request that the largest contiguous free area on the disk be obtained. In this case, the size value `CONTIG_MAX` (-1) is used instead of an actual size. These `ioctl()` codes are not supported for directories. The volume is searched for a contiguous area of free space, which is assigned to the file. If a segment of contiguous free space large enough for the request was not found, `ERROR` is returned, with `errno` set to `S_dosFsLib_NO_CONTIG_SPACE`.

When contiguous space is allocated to a file, the file remains empty, while the newly allocated space has not been initialized. The data should be then written to the file, and eventually, when all data has been written, the file is closed. When file is closed, its space is truncated to reflect the amount of data actually written to the file. This file may then be again opened and used for further I/O operations **read()** or **write()**, but it can not be guaranteed that appended data will be contiguous to the initially written data segment.

For example, the following will create a file and allocate 85 Mbytes of contiguous space:

```
fd = creat ("file", O_RDWR, 0);           /* open file          */
status = ioctl (fd, FIOCONTIG, 85*0x100000); /* get contiguous area */
if (status != OK)
    ...                                   /* do error handling  */
close (fd);                               /* close file         */
```

In contrast, the following example will create a file and allocate the largest contiguous area on the disk to it:

```
fd = creat ("file", O_RDWR, 0);           /* open file          */
status = ioctl (fd, FIOCONTIG, CONTIG_MAX); /* get contiguous area */
if (status != OK)
    ...                                   /* do error handling  */
close (fd);                               /* close file         */
```

---

**NOTE:** The FIOCONTIG operation should take place right after the file has been created, before any data is written to the file. Directories may not be allocated a contiguous disk area.

---

To determine the actual amount of contiguous space obtained when CONTIG\_MAX is specified as the size, use **fstat()** to examine the number of blocks and block size for the file.

When any file is opened, it may be checked for contiguity. Use the extended flag **DOS\_O\_CONTIG\_CHK** when calling **open()** to access an existing file which may have been allocated contiguous space. If a file is detected as contiguous, all subsequent operations on the file will not require access to the File Allocation Table, thus eliminating any disk Seek operations. The down side however is that if this option is used, **open()** will take an amount of time which is linearly proportional of the file size.

#### CHANGING, UNMOUNTING, AND SYNCHRONIZING DISKS

Buffering of disk data in RAM, synchronization of these buffers with the disk and detection of removable disk replacement are all handled by the disk cache. See reference manual on **dcacheCbio** for more details.

If a disk is physically removed, the disk cache will cause **dosFsLib** to *unmount* the volume, which will mark all currently open file descriptors as obsolete.

If a new disk is inserted, it will be automatically *mounted* on the next call to **open()** or **creat()**.

**IOCTL FUNCTIONS** The dosFs file system supports the following **ioctl()** functions. The functions listed are defined in the header **ioLib.h**. Unless stated otherwise, the file descriptor used for these functions may be any file descriptor which is opened to a file or directory on the volume or to the volume itself. There are some **ioctl()** commands, that expect a 32-bit integer result (**FIONFREE**, **FIOWHERE**, *etc.*). However, disks and files with are grater than 4GB are supported. In order to solve this problem, new **ioctl()** functions have been added to support 64-bit integer results. They have the same name as basic functions, but with suffix *64*, namely: **FIONFREE64**, **FIOWHERE64** and so on. These commands expect a pointer to a 64-bit integer, *i.e.*:

```
long long *arg ;
```

as the 3rd argument to the **ioctl()** function. If a value which is requested with a 32-bit **ioctl()** command is too large to be represented in the 32-bit variable, **ioctl()** will return **ERROR**, and **errno** will be set to **S\_dosFsLib\_32BIT\_OVERFLOW**.

#### FIODISKINIT

Re-initializes a DOS file system on the disk volume. This function calls **dosFsVolFormat()** to format the volume, so **dosFsFmtLib** must be installed for this to work. Third argument of **ioctl()** is passed as argument *opt* to **dosFsVolFormat()** routine. This routine does not perform a low level format, the physical media is expected to be already formatted. If DOS file system device has not been created yet for a particular device, only direct call to **dosFsVolFormat()** can be used.

```
fd = open ("DEV1:", O_WRONLY);
status = ioctl (fd, FIODISKINIT, DOS_OPT_BLANK);
```

#### FIODISKCHANGE

Announces a media change. No buffers flushing is performed. This function may be called from interrupt level:

```
status = ioctl (fd, FIODISKCHANGE, 0);
```

#### FIOUNMOUNT

Unmounts a disk volume. It performs the same function as **dosFsVolUnmount()**. This function must not be called from interrupt level:

```
status = ioctl (fd, FIOUNMOUNT, 0);
```

#### FIOGETNAME

Gets the file name of the file descriptor and copies it to the buffer *nameBuf*. Note that *nameBuf* must be large enough to contain the largest possible path name, which requires at least 256 bytes.

```
status = ioctl (fd, FIOGETNAME, &nameBuf );
```

#### FIORENAME

Renames the file or directory to the string *newname*:

```
fd = open( "oldname", O_RDONLY, 0 );
status = ioctl (fd, FIORENAME, "newname");
```

**FIOMOVE**

Moves the file or directory to the string *newname*:

```
fd = open( "oldname", O_RDONLY, 0 );
status = ioctl (fd, FIOMOVE, "newname");
```

**FIOSEEK**

Sets the current byte offset in the file to the position specified by *newOffset*. This function supports offsets in 32-bit value range. Use **FIOSEEK64** for larger position values:

```
status = ioctl (fd, FIOSEEK, newOffset);
```

**FIOSEEK64**

Sets the current byte offset in the file to the position specified by *newOffset*. This function supports offsets in 64-bit value range:

```
long long  newOffset;
status = ioctl (fd, FIOSEEK64, (int) & newOffset);
```

**FIOWHERE**

Returns the current byte position in the file. This is the byte offset of the next byte to be read or written. This function returns a 32-bit value. It takes no additional argument:

```
position = ioctl (fd, FIOWHERE, 0);
```

**FIOWHERE64**

Returns the current byte position in the file. This is the byte offset of the next byte to be read or written. This function returns a 64-bit value in *position*:

```
long long  position;
status = ioctl (fd, FIOWHERE64, (int) & position);
```

**FIOFLUSH**

Flushes disk cache buffers. It guarantees that any output that has been requested is actually written to the device:

```
status = ioctl (fd, FIOFLUSH, 0);
```

**FIOSYNC**

Updates the FAT copy for the passed file descriptor, then flushes and invalidates the CBIO cache buffers for the file descriptor's volume. **FIOSYNC** ensures that any outstanding output requests for the passed file descriptor are written to the device and a subsequent I/O operation will fetch data directly from the physical medium. To safely sync a volume for shutdown, all open file descriptor's should at the least be **FIOSYNC**'d by the application. Better, all open FD's should be closed by the application and the volume should be unmounted via **FIOUNMOUNT**.

```
status = ioctl (fd, FIOSYNC, 0);
```

**FIOTRUNC**

Truncates the specified file's length to *newLength* bytes. Any disk clusters which had been allocated to the file but are now unused are deallocated, and the directory entry for the file is updated to reflect the new length. Only regular files may be truncated; attempts to use **FIOTRUNC** on directories will return an error. **FIOTRUNC** may only be used to make files shorter; attempting to specify a *newLength* larger than the current size of the file produces an error (setting **errno** to **S\_dosFsLib\_INVALID\_NUMBER\_OF\_BYTES**).

```
status = ioctl (fd, FIOTRUNC, newLength);
```

**FIOTRUNC64**

Similar to **FIOTRUNC**, but can be used for files larger than 4GB.

```
long long newLength = .....;
status = ioctl (fd, FIOTRUNC, (int) & newLength);
```

**FIONREAD**

Copies to *unreadCount* the number of unread bytes in the file:

```
unsigned long unreadCount;
status = ioctl (fd, FIONREAD, &unreadCount);
```

**FIONREAD64**

Copies to *unreadCount* the number of unread bytes in the file. This function returns a 64-bit integer value:

```
long long unreadCount;
status = ioctl (fd, FIONREAD64, &unreadCount);
```

**FIONFREE**

Copies to *freeCount* the amount of free space, in bytes, on the volume:

```
unsigned long freeCount;
status = ioctl (fd, FIONFREE, &freeCount);
```

**FIONFREE64**

Copies to *freeCount* the amount of free space, in bytes, on the volume. This function can return value in 64-bit range:

```
long long freeCount;
status = ioctl (fd, FIONFREE64, &freeCount);
```

**FIOMKDIR**

Creates a new directory with the name specified as *dirName*:

```
status = ioctl (fd, FIOMKDIR, "dirName");
```

**FIORMDIR**

Removes the directory whose name is specified as *dirName*:

```
status = ioctl (fd, FIORMDIR, "dirName");
```

#### FIOLABELGET

Gets the volume label (located in root directory) and copies the string to *labelBuffer*. If the label contains `DOS_VOL_LABEL_LEN` significant characters, resulting string is not NULL terminated:

```
char    labelBuffer [DOS_VOL_LABEL_LEN];
status = ioctl (fd, FIOLABELGET, (int)labelBuffer);
```

#### FIOLABELSET

Sets the volume label to the string specified as *newLabel*. The string may consist of up to eleven ASCII characters:

```
status = ioctl (fd, FIOLABELSET, (int)"newLabel");
```

#### FIOATTRIBSET

Sets the file attribute byte in the DOS directory entry to the new value *newAttrib*. The file descriptor refers to the file whose entry is to be modified:

```
status = ioctl (fd, FIOATTRIBSET, newAttrib);
```

#### FIOCONTIG

Allocates contiguous disk space for a file or directory. The number of bytes of requested space is specified in *bytesRequested*. In general, contiguous space should be allocated immediately after the file is created:

```
status = ioctl (fd, FIOCONTIG, bytesRequested);
```

#### FIOCONTIG64

Allocates contiguous disk space for a file or directory. The number of bytes of requested space is specified in *bytesRequested*. In general, contiguous space should be allocated immediately after the file is created. This function accepts a 64-bit value:

```
long long bytesRequested;
status = ioctl (fd, FIOCONTIG64, &bytesRequested);
```

#### FIONCONTIG

Copies to *maxContigBytes* the size of the largest contiguous free space, in bytes, on the volume:

```
status = ioctl (fd, FIONCONTIG, &maxContigBytes);
```

#### FIONCONTIG64

Copies to *maxContigBytes* the size of the largest contiguous free space, in bytes, on the volume. This function returns a 64-bit value:

```
long long maxContigBytes;
status = ioctl (fd, FIONCONTIG64, &maxContigBytes);
```

#### FIOREADDIR

Reads the next directory entry. The argument *dirStruct* is a DIR directory descriptor. Normally, the `readdir()` routine is used to read a directory, rather than using the `FIOREADDIR` function directly. See `dirLib`.

```
DIR dirStruct;
fd = open ("directory", O_RDONLY);
status = ioctl (fd, FIOREADDIR, &dirStruct);
```

**FIOFSTATGET**

Gets file status information (directory entry data). The argument *statStruct* is a pointer to a stat structure that is filled with data describing the specified file. Normally, the **stat()** or **fstat()** routine is used to obtain file information, rather than using the FIOFSTATGET function directly. See **dirLib**.

```
struct stat statStruct;
fd = open ("file", O_RDONLY);
status = ioctl (fd, FIOFSTATGET, (int)&statStruct);
```

**FIOTIMESET**

Update time on a file. *arg* shall be a pointer to a **utimbuf** structure, see **utime.h**. If *arg* is value **NULL**, the current system time is used for both **actime** and **modtime** members. If *arg* is not **NULL** then the **utimbuf** structure members **actime** and **modtime** are used as passed. If **actime** is zero value, the file access time is not updated (the operation is ignored). If **modtime** is zero, the file modification time is not updated (the operation is ignored). See also **utime()**

```
struct utimbuf newTimeBuf;;
newTimeBuf.modtime = newTimeBuf.actime = fileNewTime;
fd = open ("file", O_RDONLY);
status = ioctl (fd, FIOTIMESET, (int)&newTimeBuf);
```

**FIOCHKDSK**

This function invokes the integral consistency checking. During the test, the file system will be blocked from application code access, and will emit messages describing any inconsistencies found on the disk, as well as some statistics, depending on the verbosity level in the *flags* argument. Depending on the repair permission value in *flags* argument, the inconsistencies will be repaired, and changes written to disk or only reported. Argument *flags* should be composed of bitwise or-ed verbosity level value and repair permission value. Possible repair levels are:

**DOS\_CHK\_ONLY (1)**

Only report errors, do not modify disk.

**DOS\_CHK\_REPAIR (2)**

Repair any errors found.

Possible verbosity levels are:

**DOS\_CHK\_VERB\_SILENT (0xff00)**

Do not emit any messages, except errors encountered.

**DOS\_CHK\_VERB\_1 (0x0100)**

Display some volume statistics when done testing, as well

**dpartCbio**

DOS\_CHK\_VERB\_2 (0x0200)

In addition to the above option, display path of every file, while it is being checked. This option may significantly slow down the test process.

---

**NOTE:** In environments with reduced RAM size check disk uses reserved FAT copy as temporary buffer, it can cause respectively long time of execution on a slow CPU architectures.

---

See also the reference manual **usrFsLib** for the **chkdsk()** user level utility which may be used to invoke the **FIOCHKDSK ioctl()**. The volume root directory should be opened, and the resulting file descriptor should be used:

```
int fd = open (device_name, O_RDONLY, 0);
status = ioctl (fd, FIOCHKDSK, DOS_CHK_REPAIR | DOS_CHK_VERB_1);
close (fd);
```

Any other **ioctl()** function codes are passed to the underlying **CBIO** modules for handling.

**INCLUDE FILES**

**dosFsLib.h**

**SEE ALSO**

**ioLib**, **iosLib**, **dirLib**, **usrFsLib**, **dcacheCbio**, **dpartCbio**, **dosFsFmtLib**, **dosChkLib**  
*Microsoft MS-DOS Programmer's Reference* (Microsoft Press), *Advanced MS-DOS Programming* (Ray Duncan, Microsoft Press), *VxWorks Programmer's Guide: I/O System, Local File Systems*

---

## dpartCbio

**NAME**

**dpartCbio** – generic disk partition manager

**ROUTINES**

**dpartDevCreate()** - Initialize a partitioned disk  
**dpartPartGet()** - retrieve handle for a partition

**DESCRIPTION**

This module implements a generic partition manager using the **CBIO** API (see **cbioLib**) It supports creating a separate file system device for each of its partitions.

This partition manager depends upon an external library to decode a particular disk partition table format, and report the resulting partition layout information back to this module. This module is responsible for maintaining the partition logic during operation.

When using this module with the **dcacheCbio** module, it is recommended this module be the master **CBIO** device. This module should be above the cache **CBIO** module layer. This is because the cache layer is optimized to function efficiently atop a single physical disk drive. One should call **dcacheDevCreate()** before **dpartDevCreate()**.

An implementation of the de-facto standard partition table format which is created by the MSDOS FDISK program is provided with the **usrFdiskPartLib** module, which should be used to handle PC-style partitioned hard or removable drives.

**EXAMPLE**

The following code will initialize a disk which is expected to have up to 4 partitions:

```
usrPartDiskFsInit( BLK_DEV * blkDevId )
{
    const char * devNames[] = { "/sd0a", "/sd0b", "/sd0c", "/sd0d" };
    cbioCache;
    CBIO_DEV_ID cbioParts;
    /* create a disk cache atop the entire BLK_DEV */
    cbioCache = dcacheDevCreate ( blkDevId, NULL, 0, "/sd0" );
    if (NULL == cbioCache)
    {
        return (ERROR);
    }
    /* create a partition manager with a FDISK style decoder */
    cbioParts = dpartDevCreate( cbioCache, 4, usrFdiskPartRead );
    if (NULL == cbioParts)
    {
        return (ERROR);
    }
    /* create file systems atop each partition */
    dosFsDevCreate( devNames[0], dpartPartGet( cbioParts, 0), 0x10, NONE);
    dosFsDevCreate( devNames[1], dpartPartGet( cbioParts, 1), 0x10, NONE);
    dosFsDevCreate( devNames[2], dpartPartGet( cbioParts, 2), 0x10, NONE);
    dosFsDevCreate( devNames[3], dpartPartGet( cbioParts, 3), 0x10, NONE);
}
```

Because this module complies with the CBIO programming interface on both its upper and lower layers, it is both an optional and a stackable module.

**SEE ALSO**

**dcacheLib**, **dosFsLib**, **usrFdiskPartLib**

---

## dspLib

<b>NAME</b>	<b>dspLib</b> – dsp support library
<b>ROUTINES</b>	<b>dspInit()</b> - initialize dsp support
<b>DESCRIPTION</b>	This library provides a general interface to the dsp. To activate dsp support, <b>dspInit()</b> must be called before any tasks using the dsp are spawned. This is done automatically by the root task, <b>usrRoot()</b> , in <b>usrConfig.c</b> when <b>INCLUDE_DSP</b> is defined in <b>configAll.h</b> . For information about architecture-dependent dsp routines, see the entry for <b>dspArchLib</b> .

### **VX\_DSP\_TASK** OPTION

Saving and restoring dsp registers adds to the context switch time of a task. Therefore, dsp registers are not saved and restored for every task. Only those tasks spawned with the task option **VX\_DSP\_TASK** will have dsp registers saved and restored.

---

**NOTE:** If a task does any dsp operations, it must be spawned with **VX\_DSP\_TASK**.

---

<b>INTERRUPT LEVEL</b>	DSP registers are not saved and restored for interrupt service routines connected with <b>intConnect()</b> . However, if necessary, an interrupt service routine can save and restore dsp registers by calling routines in <b>dspArchLib</b> .
------------------------	--

<b>INCLUDE FILES</b>	<b>dspLib.h</b>
----------------------	-----------------

<b>SEE ALSO</b>	<b>dspArchLib</b> , <b>dspShow</b> , <b>intConnect()</b> , <i>VxWorks Programmer's Guide: Basic OS</i>
-----------------	--

---

## dspShow

<b>NAME</b>	<b>dspShow</b> – dsp show routines
<b>ROUTINES</b>	<b>dspShowInit()</b> - initialize the dsp show facility <b>dspTaskRegsShow()</b> - print the contents of a task's dsp registers
<b>DESCRIPTION</b>	This library provides routines necessary to show a task's optional dsp context. This facility must first be installed using <b>dspShowInit()</b> . It is included automatically when <b>INCLUDE_SHOW_ROUTINES</b> and <b>INCLUDE_DSP</b> are defined in <b>configAll.h</b> .  This library enhances task information routines, such as <b>ti()</b> , to display the dsp context.
<b>INCLUDE FILES</b>	<b>dspLib.h</b>
<b>SEE ALSO</b>	<b>dspLib</b>

---

## envLib

<b>NAME</b>	<b>envLib</b> – environment variable library
<b>ROUTINES</b>	<b>envLibInit()</b> - initialize environment variable facility <b>envPrivateCreate()</b> - create a private environment <b>envPrivateDestroy()</b> - destroy a private environment <b>putenv()</b> - set an environment variable <b>getenv()</b> - get an environment variable (ANSI) <b>envShow()</b> - display the environment for a task
<b>DESCRIPTION</b>	<p>This library provides a UNIX-compatible environment variable facility. Environment variables are created or modified with a call to <b>putenv()</b>:</p> <pre>putenv ("variableName=value");</pre> <p>The value of a variable may be retrieved with a call to <b>getenv()</b>, which returns a pointer to the value string.</p> <p>Tasks may share a common set of environment variables, or they may optionally create their own private environments, either automatically when the task create hook is installed, or by an explicit call to <b>envPrivateCreate()</b>. The task must be spawned with the <b>VX_PRIVATE_ENV</b> option set to receive a private set of environment variables. Private environments created by the task creation hook inherit the values of the environment of the task that called <b>taskSpawn()</b> (since task create hooks run in the context of the calling task).</p>
<b>INCLUDE FILES</b>	<b>envLib.h</b>
<b>SEE ALSO</b>	UNIX BSD 4.3 manual entry for <b>environ(5V)</b> , * <i>American National Standard for Information Systems - * Programming Language - C, ANSI X3.159-1989: General Utilities (stdlib.h)</i>

## errnoLib

**NAME** **errnoLib** – error status library

**ROUTINES** **errnoGet()** - get the error status value of the calling task  
**errnoOfTaskGet()** - get the error status value of a specified task  
**errnoSet()** - set the error status value of the calling task  
**errnoOfTaskSet()** - set the error status value of a specified task

**DESCRIPTION** This library contains routines for setting and examining the error status values of tasks and interrupts. Most VxWorks functions return **ERROR** when they detect an error, or **NULL** in the case of functions returning pointers. In addition, they set an error status that elaborates the nature of the error.

This facility is compatible with the UNIX error status mechanism in which error status values are set in the global variable **errno**. However, in VxWorks there are many task and interrupt contexts that share common memory space and therefore conflict in their use of this global variable. VxWorks resolves this in two ways:

- (1) For tasks, VxWorks maintains the **errno** value for each context separately, and saves and restores the value of **errno** with every context switch. The value of **errno** for a non-executing task is stored in the task's TCB. Thus, regardless of task context, code can always reference or modify **errno** directly.
- (2) For interrupt service routines, VxWorks saves and restores **errno** on the interrupt stack as part of the interrupt enter and exit code provided automatically with the **intConnect()** facility. Thus, interrupt service routines can also reference or modify **errno** directly.

The **errno** facility is used throughout VxWorks for error reporting. In situations where a lower-level routine has generated an error, by convention, higher-level routines propagate the same error status, leaving **errno** with the value set at the deepest level. Developers are encouraged to use the same mechanism for application modules where appropriate.

### ERROR STATUS VALUES

An error status is a 4-byte integer. By convention, the most significant two bytes are the module number, which indicates the module in which the error occurred. The lower two bytes indicate the specific error within that module. Module number 0 is reserved for UNIX error numbers so that values from the UNIX **errno.h** header file can be set and tested without modification. Module numbers 1-500 decimal are reserved for VxWorks modules. These are defined in **vwModNum.h**. All other module numbers are available to applications.

### PRINTING ERROR STATUS VALUES

VxWorks can include a special symbol table called **statSymTbl** which **printErrno()** uses

to print human-readable error messages.

This table is created with the tool **makeStatTbl**, found in **host/hostOs/bin**. This tool reads all the.h files in a specified directory and generates a C-language file, which generates a symbol table when compiled. Each symbol consists of an error status value and its definition, which was obtained from the header file.

For example, suppose the header file **target/h/myFile.h** contains the line:

```
#define S_myFile_ERROR_TOO_MANY_COOKS      0x230003
```

The table **statSymTbl** is created by first running:

On Unix:

```
makeStatTbl target/h > statTbl.c
```

On Windows:

```
makeStatTbl target/h
```

This creates a file **statTbl.c** in the current directory, which, when compiled, generates **statSymTbl**. The table is then linked in with VxWorks. Normally, these steps are performed automatically by the makefile in **target/src/usr**.

If the user now types from the VxWorks shell:

```
-> printErrno 0x230003
```

The **printErrno()** routine would respond:

```
S_myFile_ERROR_TOO_MANY_COOKS
```

The **makeStatTbl** tool looks for error status lines of the form:

```
#define S_xxx <n>
```

where *xxx* is any string, and *n* is any number. All VxWorks status lines are of the form:

```
#define S_thisFile_MEANINGFUL_ERROR_MESSAGE 0xnnnn
```

where *thisFile* is the name of the module.

This facility is available to the user by adding header files with status lines of the appropriate forms and remaking VxWorks.

#### INCLUDE FILES

The file **vwModNum.h** contains the module numbers for every VxWorks module. The include file for each module contains the error numbers which that module can generate.

#### SEE ALSO

**printErrno()**, **makeStatTbl**, *VxWorks Programmer's Guide: Basic OS*

## etherMultiLib

<b>NAME</b>	<b>etherMultiLib</b> – a library to handle Ethernet multicast addresses
<b>ROUTINES</b>	<b>etherMultiAdd()</b> - add multicast address to a multicast address list <b>etherMultiDel()</b> - delete an Ethernet multicast address record <b>etherMultiGet()</b> - retrieve a table of multicast addresses from a driver
<b>DESCRIPTION</b>	This library manages a list of multicast addresses for network drivers. This abstracts the management of these drivers into a device-independent library.  To use this feature, include the following component: <b>INCLUDE_NETWRS_ETHERMULTILIB</b>
<b>INCLUDE FILES</b>	<b>string.h, errno.h, netinet/in.h, net/if.h, lstLib.h, etherMultiLib.h</b>

---

## eventLib

<b>NAME</b>	<b>eventLib</b> – VxWorks events library
<b>ROUTINES</b>	<b>eventReceive()</b> - wait for event(s) <b>eventSend()</b> - send event(s) <b>eventClear()</b> - clear all events for current task
<b>DESCRIPTION</b>	Events are a means of communication between tasks and interrupt routines, based on a synchronous model. Only tasks can receive events, and both tasks and ISRs can send them.  Events are similar to signals in that they are directed at one task but differ in the fact that they are synchronous in nature. Thus, the receiving task must pend when waiting for events to occur. Also, unlike signals, a handler is not needed since, when wanted events are received, the pending task continues its execution (like after a call to <b>msgQReceive()</b> or <b>semTake()</b> ).  Each task has its own events field that can be filled by having tasks (even itself) and/or ISRs sending events to the task. Each event's meaning is different for every task. Event X when received can be interpreted differently by separate tasks. Also, it should be noted that events are not accumulated. If the same event is received several times, it counts as if it were received only once. It is not possible to track how many times each event has been sent to a task.

There are some VxWorks objects that can send events when they become available. They are referred to as **resources** in the context of events. They include semaphores and message queues. For example, when a semaphore becomes free, events can be sent to a task that asked for it.

**INCLUDE FILES**    **eventLib.h**

**SEE ALSO**        **taskLib, semLib, semBLib, semCLib, semMLib, msgQLib**, *VxWorks Programmer's Guide: Basic OS*

**E**

---

## excArchLib

**NAME**            **excArchLib** – architecture-specific exception-handling facilities

**ROUTINES**        **excVecInit()** - initialize the exception/interrupt vectors  
**excConnect()** - connect a C routine to an exception vector (PowerPC)  
**excIntConnect()** - connect a C routine to an asynchronous exception vector (PowerPC, ARM)  
**excCrtConnect()** - connect a C routine to a critical exception vector (PowerPC 403)  
**excIntCrtConnect()** - connect a C routine to a critical interrupt vector (PowerPC 403)  
**excVecSet()** - set a CPU exception vector (PowerPC, ARM)  
**excVecGet()** - get a CPU exception vector (PowerPC, ARM)

**DESCRIPTION**    This library contains exception-handling facilities that are architecture dependent. For information about generic (architecture-independent) exception-handling, see the manual entry for **excLib**.

**INCLUDE FILES**    **excLib.h**

**SEE ALSO**        **excLib, dbgLib, sigLib, intLib**

---

## excLib

<b>NAME</b>	<b>excLib</b> – generic exception handling facilities
<b>ROUTINES</b>	<b>excInit()</b> - initialize the exception handling package <b>excHookAdd()</b> - specify a routine to be called with exceptions <b>excTask()</b> - handle task-level exceptions
<b>DESCRIPTION</b>	This library provides generic initialization facilities for handling exceptions. It safely traps and reports exceptions caused by program errors in VxWorks tasks, and it reports occurrences of interrupts that are explicitly connected to other handlers. For information about architecture-dependent exception handling facilities, see the manual entry for <b>excArchLib</b> .
<b>INITIALIZATION</b>	Initialization of <b>excLib</b> facilities occurs in two steps. First, the routine <b>excVecInit()</b> is called to set all vectors to the default handlers for an architecture provided by the corresponding architecture exception handling library. Since this does not involve VxWorks' kernel facilities, it is usually done early in the system start-up routine <b>usrInit()</b> in the library <b>usrConfig.c</b> with interrupts disabled.  The rest of this package is initialized by calling <b>excInit()</b> , which spawns the exception support task, <b>excTask()</b> , and creates the message queues used to communicate with it.  Exceptions or uninitialized interrupts that occur after the vectors have been initialized by <b>excVecInit()</b> , but before <b>excInit()</b> is called, cause a trap to the ROM monitor.

### NORMAL EXCEPTION HANDLING

When a program error generates an exception (such as divide by zero, or a bus or address error), the task that was executing when the error occurred is suspended, and a description of the exception is displayed on standard output. The VxWorks kernel and other system tasks continue uninterrupted. The suspended task can be examined with the usual VxWorks routines, including **ti()** for task information and **tt()** for a stack trace. It may be possible to fix the task and resume execution with **tr()**. However, tasks aborted in this way are often unsalvageable and can be deleted with **td()**.

When an interrupt that is not connected to a handler occurs, the default handler provided by the architecture-specific module displays a description of the interrupt on standard output.

### ADDITIONAL EXCEPTION HANDLING ROUTINE

The **excHookAdd()** routine adds a routine that will be called when a hardware exception occurs. This routine is called at the end of normal exception handling.

### TASK-LEVEL SUPPORT

The **excInit()** routine spawns **excTask()**, which performs special exception handling

functions that need to be done at task level. Do not suspend, delete, or change the priority of this task.

- DBGLIB** The facilities of **excLib**, including **excTask()**, are used by **dbgLib** to support breakpoints, single-stepping, and additional exception handling functions.
- SIGLIB** A higher-level, UNIX-compatible interface for hardware and software exceptions is provided by **sigLib**. If **sigvec()** is used to initialize the appropriate hardware exception/interrupt (*e.g.*, `BUS ERROR == SIGSEGV`), **excLib** will use the signal mechanism instead.
- INCLUDE FILES** **excLib.h**
- SEE ALSO** **dbgLib, sigLib, intLib**

---

## fiolib

<b>NAME</b>	<b>fiolib</b> – formatted I/O library
<b>ROUTINES</b>	<b>fiolibInit()</b> - initialize the formatted I/O support library <b>printf()</b> - write a formatted string to the standard output stream (ANSI) <b>printErr()</b> - write a formatted string to the standard error stream <b>fdprintf()</b> - write a formatted string to a file descriptor <b>sprintf()</b> - write a formatted string to a buffer (ANSI) <b>vprintf()</b> - write a string formatted with a variable argument list to standard output (ANSI) <b>vfdprintf()</b> - write a string formatted with a variable argument list to a file descriptor <b>vsprintf()</b> - write a string formatted with a variable argument list to a buffer (ANSI) <b>fiolibFormatV()</b> - convert a format string <b>fiolibRead()</b> - read a buffer <b>fiolibRdString()</b> - read a string from a file <b>sscanf()</b> - read and convert characters from an ASCII string (ANSI)
<b>DESCRIPTION</b>	<p>This library provides the basic formatting and scanning I/O functions. It includes some routines from the ANSI-compliant <b>printf()/scanf()</b> family of routines. It also includes several utility routines.</p> <p>If the floating-point format specifications <b>e</b>, <b>E</b>, <b>f</b>, <b>g</b>, and <b>G</b> are to be used with these routines, the routine <b>floatInit()</b> must be called first. If the configuration macro <b>INCLUDE_FLOATING_POINT</b> is defined, <b>floatInit()</b> is called by the root task, <b>usrRoot()</b>, in <b>usrConfig.c</b>.</p> <p>These routines do not use the buffered I/O facilities provided by the standard I/O facility. Thus, they can be invoked even if the standard I/O package has not been included. This includes <b>printf()</b>, which in most UNIX systems is part of the buffered standard I/O facilities. Because <b>printf()</b> is so commonly used, it has been implemented as an unbuffered I/O function. This allows minimal formatted I/O to be achieved without the overhead of the entire standard I/O package. For more information, see the manual entry for <b>ansiStdio</b>.</p>
<b>INCLUDE FILES</b>	<b>fiolib.h</b> , <b>stdio.h</b>
<b>SEE ALSO</b>	<b>ansiStdio</b> , <b>floatLib</b> , <i>VxWorks Programmer's Guide: I/O System</i>

---

## floatLib

**NAME** **floatLib** – floating-point formatting and scanning library

**ROUTINES** **floatInit()** - initialize floating-point I/O support

**DESCRIPTION** This library provides the floating-point I/O formatting and scanning support routines.

The floating-point formatting and scanning support routines are not directly callable; they are connected to call-outs in the **printf()/scanf()** family of functions in **fioLib**. This is done dynamically by the routine **floatInit()**, which is called by the root task, **usrRoot()**, in **usrConfig.c** when the configuration macro **INCLUDE\_FLOATING\_POINT** is defined. If this option is omitted (*i.e.*, **floatInit()** is not called), floating-point format specifications in **printf()** and **scanf()** are not supported.

**INCLUDE FILES** **math.h**

**SEE ALSO** **fioLib**

---

## fppArchLib

**NAME** **fppArchLib** – architecture-dependent floating-point coprocessor support

**ROUTINES** **fppSave()** - save the floating-point coprocessor context  
**fppRestore()** - restore the floating-point coprocessor context  
**fppProbe()** - probe for the presence of a floating-point coprocessor  
**fppTaskRegsGet()** - get the floating-point registers from a task TCB  
**fppTaskRegsSet()** - set the floating-point registers of a task

**DESCRIPTION** This library contains architecture-dependent routines to support the floating-point coprocessor. The routines **fppSave()** and **fppRestore()** save and restore all the task floating-point context information. The routine **fppProbe()** checks for the presence of the floating-point coprocessor. The routines **fppTaskRegsSet()** and **fppTaskRegsGet()** inspect and set coprocessor registers on a per-task basis.

With the exception of **fppProbe()**, the higher-level facilities in **dbgLib** and **usrLib** should be used instead of these routines. For information about architecture-independent access mechanisms, see the manual entry for **fppLib**.

**INITIALIZATION** To activate floating-point support, **fppInit()** must be called before any tasks using the coprocessor are spawned. This is done by the root task, **usrRoot()**, in **usrConfig.c**. See the manual entry for **fppLib**.

#### **x86 ARCHITECTURE**

There are two kind of floating-point contexts and set of routines for each kind. One is 108 bytes for older FPU (i80387, i80487, Pentium) and older MMX technology and **fppSave()**, **fppRestore()**, **fppRegsToCtx()**, and **fppCtxToRegs()** are used to save and restore the context, convert to or from the **FPPREG\_SET**. The other is 512 bytes for newer FPU, newer MMX technology and streaming SIMD technology (PentiumII, III, 4) and **fppXsave()**, **fppXrestore()**, **fppXregsToCtx()**, and **fppXctxToRegs()** are used to save and restore the context, convert to or from the **FPPREG\_SET**. Which to use is automatically detected by checking CPUID information in **fppArchInit()**. And **fppTaskRegsSet()** and **fppTaskRegsGet()** access the appropriate floating-point context. The bit interrogated for the automatic detection is the "Fast Save and Restore" feature flag.

#### **x86 INITIALIZATION**

To activate floating-point support, **fppInit()** must be called before any tasks using the coprocessor are spawned. If **INCLUDE\_FLOATING\_POINT** is defined in **configAll.h**, this is done by the root task, **usrRoot()**, in **usrConfig.c**.

#### **x86 VX\_FP\_TASK OPTION**

Saving and restoring floating-point registers adds to the context switch time of a task. Therefore, floating-point registers are *not* saved and restored for *every* task. Only those tasks spawned with the task option **VX\_FP\_TASK** will have floating-point state, MMX technology state, and streaming SIMD state saved and restored.

---

**NOTE:** If a task does any floating-point operations, MMX operations, and streaming SIMD operation, it must be spawned with **VX\_FP\_TASK**. It is deadly to execute any floating-point operations in a task spawned without **VX\_FP\_TASK** option, and very difficult to find. To detect that illegal/unintentional/accidental floating-point operations, a new API and mechanism is added. The mechanism is to enable or disable the FPU by toggling the TS flag in the CR0 in the new task switch hook routine - **fppArchSwitchHook()** - respecting the **VX\_FP\_TASK** option. If **VX\_FP\_TASK** option is not set in the switching-in task, the FPU is disabled. Thus the device-not-available exception will be raised if that task does any floating-point operations. This mechanism is disabled in the default. To enable, call the enabler - **fppArchSwitchHookEnable()** - with a parameter **TRUE(1)**. A parameter **FALSE(0)** disables the mechanism.

---

#### **x86 MIXING MMX AND FPU INSTRUCTIONS**

A task with **VX\_FP\_TASK** option saves and restores the FPU and MMX state when performing a context switch. Therefore, the application does not have to save or restore the FPU and MMX state if the FPU and MMX instructions are not mixed within a task. Because the MMX registers are aliased to the FPU registers, care must be taken when

making transitions between FPU instructions and MMX instructions to prevent the loss of data in the FPU and MMX registers and to prevent incoherent or unexpected result. When mixing MMX and FPU instructions within a task, follow these guidelines from Intel:

- Keep the code in separate modules, procedures, or routines.
- Do not rely on register contents across transitions between FPU and MMX code modules.
- When transitioning between MMX code and FPU code, save the MMX register state (if it will be needed in the future) and execute an EMMS instruction to empty the MMX state.
- When transitioning between FPU and MMX code, save the FPU state, if it will be needed in the future.

#### x86 MIXING SSE/SSE2 AND FPU/MMX INSTRUCTIONS

The XMM registers and the FPU/MMX registers represent separate execution environments, which has certain ramifications when executing SSE, SSE2, MMX and FPU instructions in the same task context:

- Those SSE and SSE2 instruction that operate only on the XMM registers (such as the packed and scalar floating-point instructions and the 128-bit SIMD integer instructions) can be executed in the same instruction stream with 64-bit SIMD integer or FPU instructions without any restrictions. For example, an application can perform the majority of its floating-point computations in the XMM registers, using the packed and scalar floating-point instructions, and at the same time use the FPU to perform trigonometric and other transcendental computations. Likewise, an application can perform packed 64-bit and 128-bit SIMD integer operations can be executed together without restrictions.
- Those SSE and SSE2 instructions that operate on MMX registers (such as the CVTTPS2PI, CVTTPI2PS, CVTPD2PI, CVTTPD2PI, CVTPI2PD, MOVDQ2Q, MOVQ2DQ, PADDQ, and PSUBQ instructions) can also be executed in the same instruction stream as 64-bit SIMD integer or FPU instructions, however, here they subject to the restrictions on the simultaneous use of MMX and FPU instructions, which mentioned in the previous paragraph.

#### x86 INTERRUPT LEVEL

Floating-point registers are *not* saved and restored for interrupt service routines connected with `intConnect()`. However, if necessary, an interrupt service routine can save and restore floating-point registers by calling routines in `fppALib`. See the manual entry for `intConnect()` for more information.

#### x86 EXCEPTIONS

There are six FPU exceptions that can send an exception to the CPU. They are controlled by Exception Mask bits of the Control Word register. VxWorks disables them in the default configuration. They are:

- Precision

**fppLib**

- Overflow
- Underflow
- Division by zero
- Denormalized operand
- Invalid Operation

**ARM ARCHITECTURE**

This architecture does not currently support floating-point coprocessors.

**INCLUDE FILES**    **fppLib.h**

**SEE ALSO**        **fppLib**, **intConnect()**, *Motorola MC68881/882 Floating-Point Coprocessor User's Manual*, *Intel 80960SA/SB Reference Manual*, *Intel 80960KB Programmer's Reference Manual*, *Intel 387 DX User's Manual*, *Intel Architecture Software Developer's Manual*, *Hitachi SH7750 Hardware Manual*, Gerry Kane and Joe Heinrich: *MIPS RISC Architecture Manual*

---

## fppLib

**NAME**            **fppLib** – floating-point coprocessor support library

**ROUTINES**        **fppInit()** - initialize floating-point coprocessor support

**DESCRIPTION**    This library provides a general interface to the floating-point coprocessor. To activate floating-point support, **fppInit()** must be called before any tasks using the coprocessor are spawned. This is done automatically by the root task, **usrRoot()**, in **usrConfig.c** when the configuration macro **INCLUDE\_HW\_FP** is defined.

For information about architecture-dependent floating-point routines, see the manual entry for **fppArchLib**.

The **fppShow()** routine displays coprocessor registers on a per-task basis. For information on this facility, see the manual entries for **fppShow** and **fppShow()**.

**VX\_FP\_TASK OPTION**

Saving and restoring floating-point registers adds to the context switch time of a task. Therefore, floating-point registers are not saved and restored for every task. Only those tasks spawned with the task option **VX\_FP\_TASK** will have floating-point registers saved and restored.

---

**NOTE:** If a task does any floating-point operations, it must be spawned with **VX\_FP\_TASK**.

---

<b>INTERRUPT LEVEL</b>	Floating-point registers are not saved and restored for interrupt service routines connected with <b>intConnect()</b> . However, if necessary, an interrupt service routine can save and restore floating-point registers by calling routines in <b>fppArchLib</b> .
<b>INCLUDE FILES</b>	<b>fppLib.h</b>
<b>SEE ALSO</b>	<b>fppArchLib</b> , <b>fppShow</b> , <b>intConnect()</b> , <i>VxWorks Programmer's Guide: Basic OS</i>

---

## fppShow

<b>NAME</b>	<b>fppShow</b> – floating-point show routines
<b>ROUTINES</b>	<b>fppShowInit()</b> - initialize the floating-point show facility <b>fppTaskRegsShow()</b> - print the contents of a task's floating-point registers
<b>DESCRIPTION</b>	<p>This library provides the routines necessary to show a task's optional floating-point context. To use this facility, it must first be installed using <b>fppShowInit()</b>, which is called automatically when the floating-point show facility is configured into VxWorks using either of the following methods:</p> <p>If you use the configuration header files, define <b>INCLUDE_SHOW_ROUTINES</b> in <b>config.h</b>.</p> <p>If you use the Tornado project facility, select <b>INCLUDE_HW_FP_SHOW</b>.</p> <p>This library enhances task information routines, such as <b>ti()</b>, to display the floating-point context.</p>
<b>INCLUDE FILES</b>	<b>fppLib.h</b>
<b>SEE ALSO</b>	<b>fppLib</b>

---

## ftpLib

<b>NAME</b>	<b>ftpLib</b> – File Transfer Protocol (FTP) server
<b>ROUTINES</b>	<b>ftpInit()</b> - initialize the FTP server task <b>ftpDelete()</b> - terminate the FTP server task
<b>DESCRIPTION</b>	This library implements the server side of the File Transfer Protocol (FTP), which provides remote access to the file systems available on a target. The protocol is defined in RFC 959.

This implementation supports all commands required by that specification, as well as several additional commands.

**USER INTERFACE** During system startup, the **ftpdInit()** routine creates a control connection at the predefined FTP server port which is monitored by the primary FTP task. Each FTP session established is handled by a secondary server task created as necessary. The server accepts the following commands:

HELP - List supported commands.  
USER - Verify user name.  
PASS - Verify password for the user.  
QUIT - Quit the session.  
LIST - List out contents of a directory.  
NLST - List directory contents using a concise format.  
RETR - Retrieve a file.  
STOR - Store a file.  
CWD - Change working directory.  
TYPE - Change the data representation type.  
PORT - Change the port number.  
PWD - Get the name of current working directory.  
STRU - Change file structure settings.  
MODE - Change file transfer mode.  
ALLO - Reserve sufficient storage.  
ACCT - Identify the user's account.  
PASV - Make the server listen on a port for data connection.  
NOOP - Do nothing.  
DELE - Delete a file

The **ftpdDelete()** routine will disable the FTP server until restarted. It reclaims all system resources used by the server tasks and cleanly terminates all active sessions.

To use this feature, include the following component: **INCLUDE\_FTP\_SERVER**

**INCLUDE FILES** **ftpdLib.h**

**SEE ALSO** **ftpdLib**, **netDrv**, *RFC-959 File Transfer Protocol*

---

# ftpLib

**NAME** ftpLib – File Transfer Protocol (FTP) library

**ROUTINES**

- `ftpCommand()` - send an FTP command and get the reply
- `ftpCommandEnhanced()` - send an FTP command and get the complete RFC reply code
- `ftpXfer()` - initiate a transfer via FTP
- `ftpReplyGet()` - get an FTP command reply
- `ftpReplyGetEnhanced()` - get an FTP command reply
- `ftpHookup()` - get a control connection to the FTP server on a specified host
- `ftpLogin()` - log in to a remote FTP server
- `ftpDataConnInitPassiveMode()` - initialize an FTP data connection using PASV mode
- `ftpDataConnInit()` - initialize an FTP data connection using PORT mode
- `ftpDataConnGet()` - get a completed FTP data connection
- `ftpLs()` - list directory contents via FTP
- `ftpLibDebugOptionSet()` - set the debug level of the ftp library routines
- `ftpTransientConfigSet()` - set parameters for host `FTP_TRANSIENT` responses
- `ftpTransientConfigGet()` - get parameters for host `FTP_TRANSIENT` responses
- `ftpTransientFatalInstall()` - set applet to stop FTP transient host responses

**DESCRIPTION** This library provides facilities for transferring files to and from a host via File Transfer Protocol (FTP). This library implements only the “client” side of the FTP facilities.

**FTP IN VXWORKS** For most purposes, you should access the services of **ftpLib** by means of **netDrv**, a VxWorks I/O driver that supports transparent access to remote files by means of standard I/O system calls. Before attempting to access **ftpLib** services directly, you should check whether **netDrv** already provides the same access for less trouble.

## HIGH-LEVEL INTERFACE

The routines `ftpXfer()` and `ftpReplyGet()` provide the highest level of direct interface to FTP. The routine `ftpXfer()` connects to a specified remote FTP server, logs in under a specified user name, and initiates a specified data transfer command. The routine `ftpReplyGet()` receives control reply messages sent by the remote FTP server in response to the commands sent.

## LOW-LEVEL INTERFACE

The routines `ftpHookup()`, `ftpLogin()`, `ftpDataConnInit()`, `ftpDataConnGet()`, `ftpCommand()`, `ftpCommandEnhanced()` provide the primitives necessary to create and use control and data connections to remote FTP servers. The following example shows how to use these low-level routines. It implements roughly the same function as `ftpXfer()`.

```
char *host, *user, *passwd, *acct, *dirname, *filename;  
int ctrlSock = ERROR; /* This is the control socket file descriptor */
```

```
int dataSock = ERROR; /* This is the data path socket file descriptor */
if (((ctrlSock = ftpHookup (host)) == ERROR) ||
    (ftpLogin (ctrlSock, user, passwd, acct) == ERROR) ||
    (ftpCommand (ctrlSock, "TYPE I", 0, 0, 0, 0, 0, 0) != FTP_COMPLETE) ||
    (ftpCommand (ctrlSock, "CWD %s", dirname, 0,0,0,0,0) != FTP_COMPLETE) ||
    ((dataSock = ftpDataConnInit (ctrlSock)) == ERROR) ||
    (ftpCommand (ctrlSock, "RETR %s", filename, 0,0,0,0,0) != FTP_PRELIM) ||
    ((dataSock = ftpDataConnGet (dataSock)) == ERROR))
{
    /* an error occurred; close any open sockets and return */
    if (ctrlSock != ERROR)
        close (ctrlSock);
    if (dataSock != ERROR)
        close (dataSock);
    return (ERROR);
}
```

For even lower-level access, please note that the sockets provided by **ftpHookup()** and **ftpDataConnInit()** are standard TCP/IP sockets. Developers may implement **read()**, **write()** and **select()** calls using these sockets for maximum flexibility.

To use this feature, include the following component: **INCLUDE\_FTP**

#### TUNING FOR MULTIPLE FILE ACCESS

Please note that accessing multiple files simultaneously may require increasing the memory available to the network stack. You can examine memory requirements by using **netStackSysPoolShow()** and **netStackDataPoolShow()** before opening and after closing files.

You may need to modify the following macro definitions according to your specific memory requirements:

```
NUM_64
NUM_128
NUM_256
NUM_512
NUM_1024
NUM_2048
NUM_SYS_64
NUM_SYS_128
NUM_SYS_256
NUM_SYS_512
NUM_SYS_1024
NUM_SYS_2048
```

Please also note that each concurrent file access requires three file descriptors (File, Control and Socket). The following macro definition may need modification per your application: **NUM\_FILES**

Developers are encouraged to enable the error reporting facility during debugging using the function **ftpLibDebugOptionsSet()**. The output is displayed via the logging facility.

**INCLUDE FILES**    **ftpLib.h**

**SEE ALSO**        **netDrv, logLib**

---

## **ftruncate**

**NAME**            **ftruncate** – POSIX file truncation

**ROUTINES**       **ftruncate()** - truncate a file (POSIX)

## hostLib

<b>NAME</b>	<b>hostLib</b> – host table subroutine library
<b>ROUTINES</b>	<b>hostTblInit()</b> - initialize the network host table <b>hostAdd()</b> - add a host to the host table <b>hostDelete()</b> - delete a host from the host table <b>hostGetByName()</b> - look up a host in the host table by its name <b>hostGetByAddr()</b> - look up a host in the host table by its Internet address <b>sethostname()</b> - set the symbolic name of this machine <b>gethostname()</b> - get the symbolic name of this machine
<b>DESCRIPTION</b>	<p>This library provides routines to store and access the network host database. The host table contains information regarding the known hosts on the local network. The host table (displayed with <b>hostShow()</b>) contains the Internet address, the official host name, and aliases.</p> <p>By convention, network addresses are specified in dotted (“.”) decimal notation. The library <b>inetLib</b> contains Internet address manipulation routines. Host names and aliases may contain any printable character.</p> <p>Before any of the routines in this module can be used, the library must be initialized by <b>hostTblInit()</b>. This is done automatically if <b>INCLUDE_HOST_TBL</b> is defined.</p>
<b>INCLUDE FILES</b>	<b>hostLib.h</b>
<b>SEE ALSO</b>	<b>inetLib</b>

---

## icmpShow

<b>NAME</b>	<b>icmpShow</b> – ICMP Information display routines
<b>ROUTINES</b>	<b>icmpShowInit()</b> - initialize ICMP show routines <b>icmpstatShow()</b> - display statistics for ICMP
<b>DESCRIPTION</b>	<p>This library provides routines to show ICMP related statistics.</p> <p>Interpreting these statistics requires detailed knowledge of Internet network protocols. Information on these protocols can be found in the following books:</p> <p><i>TCP/IP Illustrated Volume II, The Implementation</i>, by Richard Stevens</p> <p><i>The Design and Implementation of the 4.4 BSD UNIX Operating System</i>, by Leffler, McKusick, Karels and Quarterman</p> <p>The <b>icmpShowInit()</b> routine links the ICMP show facility into the VxWorks system. This is performed automatically if <b>INCLUDE_NET_SHOW</b> is defined.</p>
<b>SEE ALSO</b>	<b>netLib</b> , <b>netShow</b>

---

## ifIndexLib

<b>NAME</b>	<b>ifIndexLib</b> – interface index library
<b>ROUTINES</b>	<b>ifIndexLibInit()</b> - initializes library variables <b>ifIndexLibShutdown()</b> - frees library variables <b>ifIndexAlloc()</b> - return a unique interface index <b>ifIndexTest()</b> - returns true if an index has been allocated.

---

## ifLib

<b>NAME</b>	ifLib – network interface library
<b>ROUTINES</b>	<p>ifUnnumberedSet() - configure an interface to be unnumbered</p> <p>ifAddrAdd() - add an interface address for a network interface</p> <p>ifAddrSet() - set an interface address for a network interface</p> <p>ifAddrDelete() - delete an interface address for a network interface</p> <p>ifAddrGet() - get the Internet address of a network interface</p> <p>ifBroadcastSet() - set the broadcast address for a network interface</p> <p>ifBroadcastGet() - get the broadcast address for a network interface</p> <p>ifDstAddrSet() - define an address for the other end of a point-to-point link</p> <p>ifDstAddrGet() - get the Internet address of a point-to-point peer</p> <p>ifMaskSet() - define a subnet for a network interface</p> <p>ifMaskGet() - get the subnet mask for a network interface</p> <p>ifFlagChange() - change the network interface flags</p> <p>ifFlagSet() - specify the flags for a network interface</p> <p>ifFlagGet() - get the network interface flags</p> <p>ifMetricSet() - specify a network interface hop count</p> <p>ifMetricGet() - get the metric for a network interface</p> <p>ifRouteDelete() - delete routes associated with a network interface</p> <p>ifAllRoutesDelete() - delete all routes associated with a network interface</p> <p>ifunit() - map an interface name to an interface structure pointer</p> <p>ifNameToIfIndex() - returns the interface index given the interface name</p> <p>ifIndexToIfName() - returns the interface name given the interface index</p>
<b>DESCRIPTION</b>	<p>This library contains routines to configure the network interface parameters. Generally, each routine corresponds to one of the functions of the UNIX command <b>ifconfig</b>.</p> <p>To use this feature, include the following component: <b>INCLUDE_NETWRS_IFLIB</b></p>
<b>INCLUDE FILES</b>	ifLib.h
<b>SEE ALSO</b>	hostLib

---

## igmpShow

<b>NAME</b>	<b>igmpShow</b> – IGMP information display routines
<b>ROUTINES</b>	<b>igmpShowInit()</b> - initialize IGMP show routines <b>igmpstatShow()</b> - display statistics for IGMP
<b>DESCRIPTION</b>	<p>This library provides routines to show IGMP related statistics.</p> <p>Interpreting these statistics requires detailed knowledge of Internet network protocols. Information on these protocols can be found in the following books:</p> <p><i>TCP/IP Illustrated Volume II, The Implementation</i>, by Richard Stevens</p> <p><i>The Design and Implementation of the 4.4 BSD UNIX Operating System</i>, by Leffler, McKusick, Karels and Quarterman</p> <p>The <b>igmpShowInit()</b> routine links the IGMP show facility into the VxWorks system. This is performed automatically if <b>INCLUDE_NET_SHOW</b> and <b>INCLUDE_IGMP</b> are defined.</p>
<b>SEE ALSO</b>	<b>netLib</b> , <b>netShow</b>

---

## inetLib

<b>NAME</b>	<b>inetLib</b> – internet address manipulation routines
<b>ROUTINES</b>	<b>inet_addr()</b> - convert a dot notation Internet address to a long integer <b>inet_lnaof()</b> - get the local address (host number) from the Internet address <b>inet_makeaddr_b()</b> - form an Internet address from network and host numbers <b>inet_makeaddr()</b> - form an Internet address from network and host numbers <b>inet_netof()</b> - return the network number from an Internet address <b>inet_netof_string()</b> - extract the network address in dot notation <b>inet_network()</b> - convert an Internet network number from string to address <b>inet_ntoa_b()</b> - convert a network address to dot notation, store it in a buffer <b>inet_ntoa()</b> - convert a network address to dotted decimal notation <b>inet_aton()</b> - convert a network address from dot notation, store in a structure
<b>DESCRIPTION</b>	<p>This library provides routines for manipulating Internet addresses, including the UNIX BSD 4.3 <b>inet_</b> routines. It includes routines for converting between character addresses in Internet standard dotted decimal notation and integer addresses, routines for extracting the network and host portions out of an Internet address, and routines for constructing Internet addresses given the network and host address parts.</p>

All Internet addresses are returned in network order (bytes ordered from left to right). All network numbers and local address parts are returned as machine format integer values.

#### **INTERNET ADDRESSES**

Internet addresses are typically specified in dotted decimal notation or as a 4-byte number. Values specified using the dotted decimal notation take one of the following forms:

**a.b.c.d**  
**a.b.c**  
**a.b**  
**a**

If four parts are specified, each is interpreted as a byte of data and assigned, from left to right, to the four bytes of an Internet address. Note that when an Internet address is viewed as a 32-bit integer quantity on any MC68000 family machine, the bytes referred to above appear as "a.b.c.d" and are ordered from left to right.

If a three-part address is specified, the last part is interpreted as a 16-bit quantity and placed in the right-most two bytes of the network address. This makes the three-part address format convenient for specifying Class B network addresses as "128.net.host".

If a two-part address is supplied, the last part is interpreted as a 24-bit quantity and placed in the right-most three bytes of the network address. This makes the two-part address format convenient for specifying Class A network addresses as "net.host".

If only one part is given, the value is stored directly in the network address without any byte rearrangement.

Although dotted decimal notation is the default, it is possible to use the dot notation with hexadecimal or octal numbers. The base is indicated using the same prefixes as are used in C. That is, a leading 0x or 0X indicates a hexadecimal number. A leading 0 indicates an octal number. If there is no prefix, the number is interpreted as decimal.

To use this feature, include the following component: **INCLUDE\_NETWRS\_INETLIB**

**INCLUDE FILES**     **inetLib.h, inet.h**

**SEE ALSO**         UNIX BSD 4.3 manual entry for inet(3N)

---

## inflateLib

<b>NAME</b>	<b>inflateLib</b> – inflate code using public domain <b>zlib</b> functions
<b>ROUTINES</b>	<b>inflate()</b> - inflate compressed code
<b>DESCRIPTION</b>	<p>This library is used to inflate a compressed data stream, primarily for boot ROM decompression. Compressed boot ROMs contain a compressed executable in the data segment between the symbols <b>binArrayStart</b> and <b>binArrayEnd</b> (compressed data is generated by <b>deflate()</b> and <b>binToAsm</b>). The boot ROM startup code (in <b>bootInit.c</b>) calls <b>inflate()</b> to decompress the executable and then jump to it.</p> <p>This library is based on the public domain <b>zlib</b> code, which has been modified by Wind River Systems. For more information, see the <b>zlib</b> home page at <a href="http://www.gzip.org/zlib/">http://www.gzip.org/zlib/</a>.</p>

---

## intArchLib

<b>NAME</b>	<b>intArchLib</b> – architecture-dependent interrupt library
<b>ROUTINES</b>	<p><b>intLevelSet()</b> - set the interrupt level (68K, x86, ARM, SimSolaris, SimNT and SH) <b>intLock()</b> - lock out interrupts <b>intUnlock()</b> - cancel interrupt locks <b>intEnable()</b> - enable corresponding interrupt bits (MIPS, PowerPC, ARM) <b>intDisable()</b> - disable corresponding interrupt bits (MIPS, PowerPC, ARM) <b>intCRGet()</b> - read the contents of the cause register (MIPS) <b>intCRSet()</b> - write the contents of the cause register (MIPS) <b>intSRGet()</b> - read the contents of the status register (MIPS) <b>intSRSet()</b> - update the contents of the status register (MIPS) <b>intConnect()</b> - connect a C routine to a hardware interrupt <b>intHandlerCreate()</b> - construct interrupt handler for C routine (68K, x86, MIPS, SimSolaris) <b>intLockLevelSet()</b> - set current interrupt lock-out level (68K, x86, ARM, SH, SimSolaris, SimNT) <b>intLockLevelGet()</b> - get current interrupt lock-out level (68K, x86, ARM, SH, SimSolaris, SimNT) <b>intVecBaseSet()</b> - set vector (trap) base address (68K, x86, MIPS, ARM, SimSolaris, SimNT) <b>intVecBaseGet()</b> - get vector (trap) base address (68K, x86, MIPS, ARM, SimSolaris, SimNT) <b>intVecSet()</b> - set a CPU vector (trap) (68K, x86, MIPS, SH, SimSolaris, SimNT)</p>

**intArchLib**

**intVecGet()** - get an interrupt vector (68K, x86, MIPS, SH, SimSolaris, SimNT)  
**intVecTableWriteProtect()** - write-protect exception vector table (68K, x86, ARM, SimSolaris, SimNT)  
**intUninitVecSet()** - set the uninitialized vector handler (ARM)  
**intHandlerCreateI86()** - construct an interrupt handler for a C routine (x86)  
**intVecSet2()** - set a CPU vector, gate type(int/trap), and selector (x86)  
**intVecGet2()** - get a CPU vector, gate type(int/trap), and gate selector (x86)  
**intStackEnable()** - enable or disable the interrupt stack usage (x86)

**DESCRIPTION**

This library provides architecture-dependent routines to manipulate and connect to hardware interrupts. Any C language routine can be connected to any interrupt by calling **intConnect()**. Vectors can be accessed directly by **intVecSet()** and **intVecGet()**. The vector (trap) base register (if present) can be accessed by the routines **intVecBaseSet()** and **intVecBaseGet()**.

Tasks can lock and unlock interrupts by calling **intLock()** and **intUnlock()**. The lock-out level can be set and reported by **intLockLevelSet()** and **intLockLevelGet()** (68K, x86, ARM and SH only). The routine **intLevelSet()** changes the current interrupt level of the processor (68K, ARM, SimSolaris, and SH).

---

**WARNING:** Do not call VxWorks system routines with interrupts locked. Violating this rule may re-enable interrupts unpredictably.

---

**INTERRUPT VECTORS AND NUMBERS**

Most of the routines in this library take an interrupt vector as a parameter, which is generally the byte offset into the vector table. Macros are provided to convert between interrupt vectors and interrupt numbers:

**IVEC\_TO\_INUM** (intVector)  
 converts a vector to a number.

**INUM\_TO\_IVEC** (intNumber)  
 converts a number to a vector.

**TRAPNUM\_TO\_IVEC** (trapNumber)  
 converts a trap number to a vector.

**EXAMPLE**

To switch between one of several routines for a particular interrupt, the following code fragment is one alternative:

```
vector = INUM_TO_IVEC(some_int_vec_num);
oldfunc = intVecGet (vector);
newfunc = intHandlerCreate (routine, parameter);
intVecSet (vector, newfunc);
...
intVecSet (vector, oldfunc); /* use original routine */
...
intVecSet (vector, newfunc); /* reconnect new routine */
```

**INCLUDE FILES**    **iv.h, intLib.h**

**SEE ALSO**        **intLib**

---

## intLib

**NAME**            **intLib** – architecture-independent interrupt subroutine library

**ROUTINES**        **intContext()** - determine if the current state is in interrupt or task context  
**intCount()** - get the current interrupt nesting depth

**DESCRIPTION**    This library provides generic routines for interrupts. Any C language routine can be connected to any interrupt (trap) by calling **intConnect()**, which resides in **intArchLib**. The **intCount()** and **intContext()** routines are used to determine whether the CPU is running in an interrupt context or in a normal task context. For information about architecture-dependent interrupt handling, see the manual entry for **intArchLib**.

**INCLUDE FILES**    **intLib.h**

**SEE ALSO**        **intArchLib**, *VxWorks Programmer's Guide: Basic OS*

---

## ioLib

**NAME**            **ioLib** – I/O interface library

**ROUTINES**        **creat()** - create a file  
**open()** - open a file  
**unlink()** - delete a file (POSIX)  
**remove()** - remove a file (ANSI)  
**close()** - close a file  
**rename()** - change the name of a file  
**read()** - read bytes from a file or device  
**write()** - write bytes to a file  
**ioctl()** - perform an I/O control function  
**lseek()** - set a file read/write pointer  
**ioDefPathSet()** - set the current default path  
**ioDefPathGet()** - get the current default path  
**chdir()** - set the current default path  
**getcwd()** - get the current default path (POSIX)

**ioLib**

**getwd()** - get the current default path

**ioGlobalStdSet()** - set the file descriptor for global standard input/output/error

**ioGlobalStdGet()** - get the file descriptor for global standard input/output/error

**ioTaskStdSet()** - set the file descriptor for task standard input/output/error

**ioTaskStdGet()** - get the file descriptor for task standard input/output/error

**isatty()** - return whether the underlying driver is a *tty* device

**DESCRIPTION**

This library contains the interface to the basic I/O system. It includes:

Interfaces to the seven basic driver-provided functions: **creat()**, **remove()**, **open()**, **close()**, **read()**, **write()**, and **ioctl()**.

Interfaces to several file system functions, including **rename()** and **lseek()**.

Routines to set and get the current working directory.

Routines to assign task and global standard file descriptors.

**FILE DESCRIPTORS**

At the basic I/O level, files are referred to by a file descriptor. A file descriptor is a small integer returned by a call to **open()** or **creat()**. The other basic I/O calls take a file descriptor as a parameter to specify the intended file.

Three file descriptors are reserved and have special meanings:

0 (**STD\_IN**) - standard input

1 (**STD\_OUT**) - standard output

2 (**STD\_ERR**) - standard error output

VxWorks allows two levels of redirection. First, there is a global assignment of the three standard file descriptors. By default, new tasks use this global assignment. The global assignment of the three standard file descriptors is controlled by the routines **ioGlobalStdSet()** and **ioGlobalStdGet()**.

Second, individual tasks may override the global assignment of these file descriptors with their own assignments that apply only to that task. The assignment of task-specific standard file descriptors is controlled by the routines **ioTaskStdSet()** and **ioTaskStdGet()**.

**INCLUDE FILES**

**ioLib.h**

**SEE ALSO**

**iosLib**, **ansiStdio**, *VxWorks Programmer's Guide: I/O System*

---

## iosLib

<b>NAME</b>	<b>iosLib</b> – I/O system library
<b>ROUTINES</b>	<b>iosInit()</b> - initialize the I/O system <b>iosDrvInstall()</b> - install an I/O driver <b>iosDrvRemove()</b> - remove an I/O driver <b>iosDevAdd()</b> - add a device to the I/O system <b>iosDevDelete()</b> - delete a device from the I/O system <b>iosDevFind()</b> - find an I/O device in the device list <b>iosFdValue()</b> - validate an open file descriptor and return the driver-specific value
<b>DESCRIPTION</b>	<p>This library is the driver-level interface to the I/O system. Its primary purpose is to route user I/O requests to the proper drivers, using the proper parameters. To do this, <b>iosLib</b> keeps tables describing the available drivers (<i>e.g.</i>, names, open files).</p> <p>The I/O system should be initialized by calling <b>iosInit()</b>, before calling any other routines in <b>iosLib</b>. Each driver then installs itself by calling <b>iosDrvInstall()</b>. The devices serviced by each driver are added to the I/O system with <b>iosDevAdd()</b>.</p> <p>The I/O system is described more fully in the <i>I/O System</i> chapter of the <i>Programmer's Guide</i>.</p>
<b>INCLUDE FILES</b>	<b>iosLib.h</b>
<b>SEE ALSO</b>	<b>intLib</b> , <b>ioLib</b> , <i>VxWorks Programmer's Guide: I/O System</i>

---

## iosShow

<b>NAME</b>	<b>iosShow</b> – I/O system show routines
<b>ROUTINES</b>	<b>iosShowInit()</b> - initialize the I/O system show facility <b>iosDrvShow()</b> - display a list of system drivers <b>iosDevShow()</b> - display the list of devices in the system <b>iosFdShow()</b> - display a list of file descriptor names in the system
<b>DESCRIPTION</b>	<p>This library contains I/O system information display routines.</p> <p>The routine <b>iosShowInit()</b> links the I/O system information show facility into the VxWorks system. It is called automatically when <b>INCLUDE_SHOW_ROUTINES</b> is defined in <b>configAll.h</b>.</p>
<b>SEE ALSO</b>	<b>intLib</b> , <b>ioLib</b> , <i>VxWorks Programmer's Guide: I/O System</i> , <b>windsh</b> , <i>Tornado User's Guide: Shell</i>

---

## ipFilterLib

<b>NAME</b>	<b>ipFilterLib</b> – IP filter hooks library
<b>ROUTINES</b>	<b>ipFilterLibInit()</b> - initialize IP filter facility <b>ipFilterHookAdd()</b> - add a routine to receive all internet protocol packets <b>ipFilterHookDelete()</b> - delete a IP filter hook routine
<b>DESCRIPTION</b>	<p>This library provides utilities that give direct access to IP packets. You can examine or process incoming raw IP packets using the hooks you installed with <b>ipFilterHookAdd()</b>. Using a filter hook, you can build IP traffic monitoring and testing tools.</p> <p>However, you should not use an IP filter hook as a standard means to provide network access to an application. The filter hook lets you see, process, and even consume packets before their intended recipients have seen the packets. For most network applications, this is too much responsibility. Thus, most network applications should access the network access through the higher-level socket interface provided by <b>sockLib</b>.</p> <p>The <b>ipFilterLibInit()</b> routine links the IP filtering facility into the VxWorks system. This is performed automatically if <b>INCLUDE_IP_FILTER</b> is defined.</p>

### VXWORKS AE PROTECTION DOMAINS

Under VxWorks AE, you can call **ipFilterHookAdd()** from within the kernel protection domain only, and the function referenced in the *ipFilterHook* parameter must reside in the kernel protection domain. This restriction does not apply to non-AE versions of VxWorks.

---

## ipProto

<b>NAME</b>	<b>ipProto</b> – an interface between the BSD IP protocol and the MUX
<b>ROUTINES</b>	<b>ipAttach()</b> - a generic attach routine for the TCP/IP network stack <b>ipDetach()</b> - a generic detach routine for the TCP/IP network stack
<b>DESCRIPTION</b>	<p>This library provides an interface between the Berkeley protocol stack and the MUX interface for both NPT and END devices. The <b>ipAttach()</b> routine binds the IP protocol to a specific device. It is called automatically during network initialization if <b>INCLUDE_END</b> is defined. The <b>ipDetach()</b> routine removes an existing binding to an END device.</p> <hr/> <p><b>NOTE:</b> The library can only transmit data to link-level destination addresses less than or equal to 64 bytes in length.</p> <hr/>
<b>INCLUDE FILES</b>	<b>end.h, muxLib.h, etherMultiLib.h, sys/ioctl.h</b>

---

## kernelLib

<b>NAME</b>	<b>kernelLib</b> – VxWorks kernel library
<b>ROUTINES</b>	<b>kernelInit()</b> - initialize the kernel <b>kernelVersion()</b> - return the kernel revision string <b>kernelTimeSlice()</b> - enable round-robin selection
<b>DESCRIPTION</b>	The VxWorks kernel provides tasking control services to an application. The libraries <b>kernelLib</b> , <b>taskLib</b> , <b>semLib</b> , <b>tickLib</b> , and <b>wdLib</b> comprise the kernel functionality. This library is the interface to the VxWorks kernel initialization, revision information, and scheduling control.

### KERNEL INITIALIZATION

The kernel must be initialized before any other kernel operation is performed. Normally kernel initialization is taken care of by the system configuration code in **usrInit()** in **usrConfig.c**.

Kernel initialization consists of the following:

- (1) Defining the starting address and size of the system memory partition. The **malloc()** routine uses this partition to satisfy memory allocation requests of other facilities in VxWorks.
- (2) Allocating the specified memory size for an interrupt stack. Interrupt service routines will use this stack unless the underlying architecture does not support a separate interrupt stack, in which case the service routine will use the stack of the interrupted task.
- (3) Specifying the interrupt lock-out level. VxWorks will not exceed the specified level during any operation. The lock-out level is normally defined to mask the highest priority possible. However, in situations where extremely low interrupt latency is required, the lock-out level may be set to ensure timely response to the interrupt in question. Interrupt service routines handling interrupts of priority greater than the interrupt lock-out level may not call any VxWorks routine.

Once the kernel initialization is complete, a root task is spawned with the specified entry point and stack size. The root entry point is normally **usrRoot()** of the **usrConfig.c** module. The remaining VxWorks initialization takes place in **usrRoot()**.

### ROUND-ROBIN SCHEDULING

Round-robin scheduling allows the processor to be shared fairly by all tasks of the same priority. Without round-robin scheduling, when multiple tasks of equal priority must share the processor, a single non-blocking task can usurp the processor until preempted by a task of higher priority, thus never giving the other equal-priority tasks a chance to run.

**K**

Round-robin scheduling is disabled by default. It can be enabled or disabled with the routine **kernelTimeSlice()**, which takes a parameter for the “time slice” (or interval) that each task will be allowed to run before relinquishing the processor to another equal-priority task. If the parameter is zero, round-robin scheduling is turned off. If round-robin scheduling is enabled and preemption is enabled for the executing task, the system tick handler will increment the task’s time-slice count. When the specified time-slice interval is completed, the system tick handler clears the counter and the task is placed at the tail of the list of tasks at its priority. New tasks joining a given priority group are placed at the tail of the group with a run-time counter initialized to zero.

Enabling round-robin scheduling does not affect the performance of task context switches, nor is additional memory allocated.

If a task blocks or is preempted by a higher priority task during its interval, it’s time-slice count is saved and then restored when the task is eligible for execution. In the case of preemption, the task will resume execution once the higher priority task completes, assuming no other task of a higher priority is ready to run. For the case when the task blocks, it is placed at the tail of the list of tasks at its priority. If preemption is disabled during round-robin scheduling, the time-slice count of the executing task is not incremented.

Time-slice counts are accrued against the task that is executing when a system tick occurs regardless of whether the task has executed for the entire tick interval. Due to preemption by higher priority tasks or ISRs stealing CPU time from the task, scenarios exist where a task can execute for less or more total CPU time than it’s allotted time slice.

**INCLUDE FILES**     **kernelLib.h**

**SEE ALSO**            **taskLib, intLib, VxWorks Programmer’s Guide: Basic OS**

---

## ledLib

**NAME** ledLib – line-editing library

**ROUTINES** ledOpen() - create a new line-editor ID  
ledClose() - discard the line-editor ID  
ledRead() - read a line with line-editing  
ledControl() - change the line-editor ID parameters

**DESCRIPTION** This library provides a line-editing layer on top of a *tty* device. The shell uses this interface for its history-editing features.

The shell history mechanism is similar to the UNIX Korn shell history facility, with a built-in line-editor similar to UNIX **vi** that allows previously typed commands to be edited. The command **h()** displays the 20 most recent commands typed into the shell; old commands fall off the top as new ones are entered.

To edit a command, type **ESC** to enter edit mode, and use the commands listed below. The **ESC** key switches the shell to edit mode. The **RETURN** key always gives the line to the shell from either editing or input mode.

The following list is a summary of the commands available in edit mode.

Movement and search commands:

<i>n</i> G	- Go to command number <i>n</i> .
/s	- Search for string <i>s</i> backward in history.
?s	- Search for string <i>s</i> forward in history.
n	- Repeat last search.
N	- Repeat last search in opposite direction.
<i>nk</i>	- Get <i>n</i> th previous shell command in history.
<i>n-</i>	- Same as “ <i>k</i> ”.
<i>nj</i>	- Get <i>n</i> th next shell command in history.
<i>n+</i>	- Same as “ <i>j</i> ”.
<i>nh</i>	- Move left <i>n</i> characters.
CTRL-H	- Same as “ <i>h</i> ”.
<i>nl</i>	- Move right <i>n</i> characters.
f SPACEfP	- Same as “ <i>l</i> ”.
<i>nw</i>	- Move <i>n</i> words forward.
<i>nW</i>	- Move <i>n</i> blank-separated words forward.
<i>ne</i>	- Move to end of the <i>n</i> th next word.
<i>nE</i>	- Move to end of the <i>n</i> th next blank-separated word.
<i>nb</i>	- Move back <i>n</i> words.

- n*B - Move back *n* blank-separated words.
- f**c* - Find character *c*, searching forward.
- F**c* - Find character *c*, searching backward.
- ^ - Move cursor to first non-blank character in line.
- \$ - Go to end of line.
- 0 - Go to beginning of line.

Insert commands (input is expected until an ESC is typed):

- a* - Append.
- A* - Append at end of line.
- c* *f*1SPACE*f**P* - Change character.
- cl* - Change character.
- cw* - Change word.
- cc* - Change entire line.
- c*\$ - Change everything from cursor to end of line.
- C* - Same as "*c*\$".
- S* - Same as "*cc*".
- i* - Insert.
- I* - Insert at beginning of line.
- R* - Type over characters.

Editing commands:

- nrc* - Replace the following *n* characters with *c*.
- nx* - Delete *n* characters starting at cursor.
- nX* - Delete *n* characters to the left of the cursor.
- d* *f*1SPACE*f**P* - Delete character.
- dl* - Delete character.
- dw* - Delete word.
- dd* - Delete entire line.
- d*\$ - Delete everything from cursor to end of line.
- D* - Same as "*d*\$".
- p* - Put last deletion after the cursor.
- P* - Put last deletion before the cursor.
- u* - Undo last command.
- ~ - Toggle case, lower to upper or vice versa.

Special commands:

CTRL-U - Delete line and leave edit mode.  
 CTRL-L - Redraw line.  
 CTRL-D - Complete symbol name.  
 f1RETURNfP - Give line to shell and leave edit mode.

The default value for *n* is 1.

**DEFICIENCIES** Since the shell toggles between raw mode and line mode, type-ahead can be lost. The **ESC**, redraw, and non-printable characters are built-in. The **EOF**, backspace, and line-delete are not imported well from **tyLib**. Instead, **tyLib** should supply and/or support these characters via **ioctl()**.

Some commands do not take counts as users might expect. For example, “*ni*” will not insert whatever was entered *n* times.

**INCLUDE FILES** **ledLib.h**

**SEE ALSO** *VxWorks Programmer’s Guide: Shell*

---

## loadLib

**NAME** **loadLib** – object module loader

**ROUTINES** **loadModule()** - load an object module into memory  
**loadModuleAt()** - load an object module into memory

**DESCRIPTION** This library provides a generic object module loading facility. Any supported format files may be loaded into memory, relocated properly, their external references resolved, and their external definitions added to the system symbol table for use by other modules and from the shell. Modules may be loaded from any I/O stream which allows repositioning of the pointer. This includes **netDrv**, NFS, or local file devices. It does not include sockets.

**EXAMPLE**

```
fdX = open ("/devX/objFile", O_RDONLY);
loadModule (fdX, LOAD_ALL_SYMBOLS);
close (fdX);
```

This code fragment would load the object file “objFile” located on device **/devX/** into memory which would be allocated from the system memory pool. All external and static definitions from the file would be added to the system symbol table.

This could also have been accomplished from the shell, by typing:

```
-> ld (1) </devX/objFile
```

**INCLUDE FILE**      **loadLib.h**

**SEE ALSO**          **usrLib, symLib, memLib, VxWorks Programmer's Guide: Basic OS**

---

## loginLib

**NAME**              **loginLib** – user login/password subroutine library

**ROUTINES**        **loginInit()** - initialize the login table  
**loginUserAdd()** - add a user to the login table  
**loginUserDelete()** - delete a user entry from the login table  
**loginUserVerify()** - verify a user name and password in the login table  
**loginUserShow()** - display the user login table  
**loginPrompt()** - display a login prompt and validate a user entry  
**loginStringSet()** - change the login string  
**loginEncryptInstall()** - install an encryption routine  
**loginDefaultEncrypt()** - default password encryption routine

**DESCRIPTION**    This library provides a login/password facility for network access to the VxWorks shell. When installed, it requires a user name and password match to gain access to the VxWorks shell from **rlogin** or **telnet**. Therefore VxWorks can be used in secure environments where access must be restricted.

Routines are provided to prompt for the user name and password, and verify the response by looking up the name/password pair in a login user table. This table contains a list of user names and encrypted passwords that will be allowed to log in to the VxWorks shell remotely. Routines are provided to add, delete, and access the login user table. The list of user names can be displayed with **loginUserShow()**.

**INSTALLATION**    The login security feature is initialized by the root task, **usrRoot()**, in **usrConfig.c**, if the configuration macro **INCLUDE\_SECURITY** is defined. Defining this macro also adds a single default user to the login table. The default user and password are defined as **LOGIN\_USER\_NAME** and **LOGIN\_PASSWORD**. These can be set to any desired name and password. More users can be added by making additional calls to **loginUserAdd()**. If **INCLUDE\_SECURITY** is not defined, access to VxWorks will not be restricted and secure.

The name/password pairs are added to the table by calling **loginUserAdd()**, which takes the name and an encrypted password as arguments. The VxWorks host tool **vxencrypt** is used to generate the encrypted form of a password. For example, to add a user name of

“fred” and password of “flintstone”, first run **vxencrypt** on the host to find the encryption of “flintstone” as follows:

```
% vxencrypt
please enter password: flintstone
encrypted password is ScebRezb9c
```

Then invoke the routine **loginUserAdd()** in VxWorks:

```
loginUserAdd ("fred", "ScebRezb9c");
```

This can be done from the shell, a start-up script, or application code.

#### LOGGING IN

When the login security facility is installed, every attempt to **rlogin** or **telnet** to the VxWorks shell will first prompt for a user name and password.

```
% rlogin target
VxWorks login: fred
Password: flintstone
->
```

The delay in prompting between unsuccessful logins is increased linearly with the number of attempts, in order to slow down password-guessing programs.

#### ENCRYPTION ALGORITHM

This library provides a simple default encryption routine, **loginDefaultEncrypt()**. This algorithm requires that passwords be at least 8 characters and no more than 40 characters.

The routine **loginEncryptInstall()** allows a user-specified encryption function to be used instead of the default.

**INCLUDE FILES**     **loginLib.h**

**SEE ALSO**         **shellLib**, **vxencrypt**, *VxWorks Programmer's Guide: Shell*

---

## logLib

**NAME** logLib – message logging library

**ROUTINES** **logInit()** - initialize message logging library  
**logMsg()** - log a formatted error message  
**logFdSet()** - set the primary logging file descriptor  
**logFdAdd()** - add a logging file descriptor  
**logFdDelete()** - delete a logging file descriptor  
**logTask()** - message-logging support task

**DESCRIPTION** This library handles message logging. It is usually used to display error messages on the system console, but such messages can also be sent to a disk file or printer.

The routines **logMsg()** and **logTask()** are the basic components of the logging system. The **logMsg()** routine has the same calling sequence as **printf()**, but instead of formatting and outputting the message directly, it sends the format string and arguments to a message queue. The task **logTask()** waits for messages on this message queue. It formats each message according to the format string and arguments in the message, prepends the ID of the sender, and writes it on one or more file descriptors that have been specified as logging output streams (by **logInit()** or subsequently set by **logFdSet()** or **logFdAdd()**).

### USE IN INTERRUPT SERVICE ROUTINES

Because **logMsg()** does not directly cause output to I/O devices, but instead simply writes to a message queue, it can be called from an ISR as well as from tasks. Normal I/O, such as **printf()** output to a serial port, cannot be done from an ISR.

### DEFERRED LOGGING

Print formatting is performed within the context of **logTask()**, rather than the context of the task calling **logMsg()**. Since formatting can require considerable stack space, this can reduce stack sizes for tasks that only need to do I/O for error output.

However, this also means that the arguments to **logMsg()** are not interpreted at the time of the call to **logMsg()**, but rather are interpreted at some later time by **logTask()**. This means that the arguments to **logMsg()** should not be pointers to volatile entities. For example, pointers to dynamic or changing strings and buffers should not be passed as arguments to be formatted. Thus the following would not give the desired results:

```
doLog (which)
{
    char string [100];
    strcpy (string, which ? "hello" : "goodbye");
    ...
    logMsg (string);
}
```

By the time `logTask()` formats the message, the stack frame of the caller may no longer exist and the pointer `string` may no longer be valid. On the other hand, the following is correct since the string pointer passed to the `logTask()` always points to a static string:

```
doLog (which)
{
    char *string;
    string = which ? "hello" : "goodbye";
    ...
    logMsg (string);
}
```

<b>INITIALIZATION</b>	To initialize the message logging facilities, the routine <code>logInit()</code> must be called before calling any other routine in this module. This is done by the root task, <code>usrRoot()</code> , in <code>usrConfig.c</code> .
<b>INCLUDE FILES</b>	<code>logLib.h</code>
<b>SEE ALSO</b>	<code>msgQLib</code> , <i>VxWorks Programmer's Guide: I/O System</i>

---

## IstLib

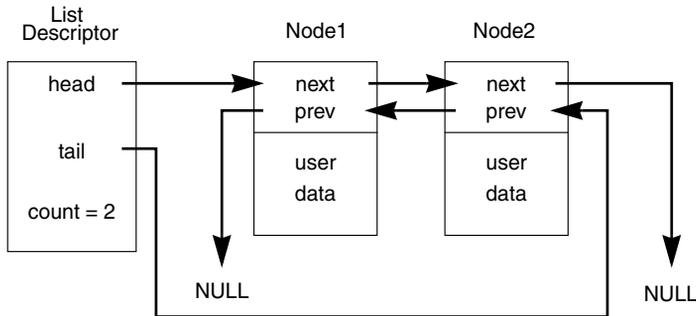
<b>NAME</b>	<code>IstLib</code> – doubly linked list subroutine library
<b>ROUTINES</b>	<p><code>IstLibInit()</code> - initializes <code>IstLib</code> module</p> <p><code>IstInit()</code> - initialize a list descriptor</p> <p><code>IstAdd()</code> - add a node to the end of a list</p> <p><code>IstConcat()</code> - concatenate two lists</p> <p><code>IstCount()</code> - report the number of nodes in a list</p> <p><code>IstDelete()</code> - delete a specified node from a list</p> <p><code>IstExtract()</code> - extract a sublist from a list</p> <p><code>IstFirst()</code> - find first node in list</p> <p><code>IstGet()</code> - delete and return the first node from a list</p> <p><code>IstInsert()</code> - insert a node in a list after a specified node</p> <p><code>IstLast()</code> - find the last node in a list</p> <p><code>IstNext()</code> - find the next node in a list</p> <p><code>IstNth()</code> - find the Nth node in a list</p> <p><code>IstPrevious()</code> - find the previous node in a list</p> <p><code>IstNStep()</code> - find a list node <i>nStep</i> steps away from a specified node</p> <p><code>IstFind()</code> - find a node in a list</p> <p><code>IstFree()</code> - free up a list</p>

**DESCRIPTION**

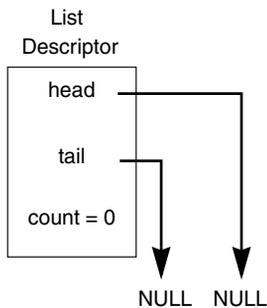
This subroutine library supports the creation and maintenance of a doubly linked list. The user supplies a list descriptor (type LIST) that will contain pointers to the first and last nodes in the list, and a count of the number of nodes in the list. The nodes in the list can be any user-defined structure, but they must reserve space for two pointers as their first elements. Both the forward and backward chains are terminated with a NULL pointer.

The linked-list library simply manipulates the linked-list data structures; no kernel functions are invoked. In particular, linked lists by themselves provide no task synchronization or mutual exclusion. If multiple tasks will access a single linked list, that list must be guarded with some mutual-exclusion mechanism (*e.g.*, a mutual-exclusion semaphore).

**NON-EMPTY LIST**



**EMPTY LIST**



**INCLUDE FILES**

IstLib.h

---

## m2IcmpLib

- NAME** **m2IcmpLib** – MIB-II ICMP-group API for SNMP Agents
- ROUTINES** **m2IcmpInit()** - initialize MIB-II ICMP-group access  
**m2IcmpGroupInfoGet()** - get the MIB-II ICMP-group global variables  
**m2IcmpDelete()** - delete all resources used to access the ICMP group
- DESCRIPTION** This library provides MIB-II services for the ICMP group. It provides routines to initialize the group, and to access the group scalar variables. For a broader description of MIB-II services, see the manual entry for **m2Lib**.

To use this feature, include the following component: **INCLUDE\_MIB2\_ICMP**

### USING THIS LIBRARY

This library can be initialized and deleted by calling the routines **m2IcmpInit()** and **m2IcmpDelete()** respectively, if only the ICMP group's services are needed. If full MIB-II support is used, this group and all other groups can be initialized and deleted by calling **m2Init()** and **m2Delete()**.

The group scalar variables are accessed by calling **m2IcmpGroupInfoGet()** as follows:

```
M2_ICMP   icmpVars;
if (m2IcmpGroupInfoGet (&icmpVars) == OK)
    /* values in icmpVars are valid */
```

- INCLUDE FILES** **m2Lib.h**
- SEE ALSO** **m2Lib**, **m2IfLib**, **m2IpLib**, **m2TcpLib**, **m2SysLib**

---

## m2IfLib

- NAME** **m2IfLib** – MIB-II interface-group API for SNMP agents
- ROUTINES** **m2IfAlloc()** - allocate the structure for the interface table  
**m2IfFree()** - free an interface data structure  
**m2IfGenericPacketCount()** - increment the interface packet counters  
**m2If8023PacketCount()** - increment the packet counters for an 802.3 device  
**m2IfCounterUpdate()** - increment interface counters  
**m2IfVariableUpdate()** - update the contents of an interface non-counter object  
**m2IfPktCountRtnInstall()** - install an interface packet counter routine  
**m2IfCtrUpdateRtnInstall()** - install an interface counter update routine

**m2IfVarUpdateRtnInstall()** - install an interface variable update routine  
**m2IfInit()** - initialize MIB-II interface-group routines  
**m2IfTableUpdate()** - insert or remove an entry in the **ifTable**  
**rcvEtherAddrGet()** - populate the *rcvAddr* fields for the **ifRcvAddressTable**  
**rcvEtherAddrAdd()** - add a physical address into the linked list  
**m2IfTblEntryGet()** - get a MIB-II interface-group table entry  
**m2IfDefaultValsGet()** - get the default values for the counters  
**m2IfCommonValsGet()** - get the common values  
**m2IfTblEntrySet()** - set the state of a MIB-II interface entry to UP or DOWN  
**m2IfGroupInfoGet()** - get the MIB-II interface-group scalar variables  
**m2IfStackTblUpdate()** - update the relationship between the sub-layers  
**stackEntryIsTop()** - test if an **ifStackTable** interface has no layers above  
**stackEntryIsBottom()** - test if an interface has no layers beneath it  
**m2IfStackEntryGet()** - get a MIB-II interface-group table entry  
**m2IfStackEntrySet()** - modify the status of a relationship  
**m2IfRcvAddrEntryGet()** - get the *rcvAddress* table entries for a given address  
**m2IfRcvAddrEntrySet()** - modify the entries of the **rcvAddressTable**  
**m2IfDelete()** - delete all resources used to access the interface group  
**nextIndex()** - the comparison routine for the AVL tree

**DESCRIPTION** This library provides MIB-II services for the interface group. It provides routines to initialize the group, access the group scalar variables, read the table interfaces and change the state of the interfaces. For a broader description of MIB-II services, see the manual entry for **m2Lib**.

To use this feature, include the following component: **INCLUDE\_MIB2\_IF**

#### USING THIS LIBRARY

This library can be initialized and deleted by calling **m2IfInit()** and **m2IfDelete()** respectively, if only the interface group's services are needed. If full MIB-II support is used, this group and all other groups can be initialized and deleted by calling **m2Init()** and **m2Delete()**.

The interface group supports the Simple Network Management Protocol (SNMP) concept of traps, as specified by RFC 1215. The traps supported by this group are "link up" and "link down." This library enables an application to register a hook routine and an argument. This hook routine can be called by the library when a "link up" or "link down" condition is detected. The hook routine must have the following prototype:

```
void TrapGenerator (int trapType, /* M2_LINK_DOWN_TRAP or M2_LINK_UP_TRAP */
                   int interfaceIndex,
                   void * myPrivateArg);
```

The trap routine and argument can be specified at initialization time as input parameters to the routine **m2IfInit()** or to the routine **m2Init()**.

The interface-group global variables can be accessed as follows:

```
M2_INTERFACE    ifVars;
if (m2IfGroupInfoGet (&ifVars) == OK)
    /* values in ifVars are valid */
```

An interface table entry can be retrieved as follows:

```
M2_INTERFACETBL interfaceEntry;
/* Specify zero as the index to get the first entry in the table */
interfaceEntry.ifIndex = 2;    /* Get interface with index 2 */
if (m2IfTblEntryGet (M2_EXACT_VALUE, &interfaceEntry) == OK)
    /* values in interfaceEntry are valid */
```

An interface entry operational state can be changed as follows:

```
M2_INTERFACETBL ifEntryToSet;
ifEntryToSet.ifIndex = 2; /* Select interface with index 2    */
                        /* MIB-II value to set the interface */
                        /* to the down state.                */
ifEntryToSet.ifAdminStatus = M2_ifAdminStatus_down;
if (m2IfTblEntrySet (&ifEntryToSet) == OK)
    /* Interface is now in the down state */
```

INCLUDE FILES    m2Lib.h

SEE ALSO        m2Lib, m2SysLib, m2IpLib, m2IcmpLib, m2UdpLib, m2TcpLib

---

## m2Igmpp

NAME            m2Igmpp – helper file for igmp Mib

ROUTINES        No Callable Routines.

DESCRIPTION     This library provides an interface between the Berkeley multicast code and the IGMP Mib code

INCLUDE FILES    m2Lib.h

---

## m2IpLib

**NAME** m2IpLib – MIB-II IP-group API for SNMP agents

**ROUTINES**

- m2IpInit()** - initialize MIB-II IP-group access
- m2IpGroupInfoGet()** - get the MIB-II IP-group scalar variables
- m2IpGroupInfoSet()** - set MIB-II IP-group variables to new values
- m2IpAddrTblEntryGet()** - get an IP MIB-II address entry
- m2IpAtransTblEntryGet()** - get a MIB-II ARP table entry
- m2IpAtransTblEntrySet()** - add, modify, or delete a MIB-II ARP entry
- m2IpRouteTblEntryGet()** - get a MIB-2 routing table entry
- m2IpRouteTblEntrySet()** - set a MIB-II routing table entry
- m2IpDelete()** - delete all resources used to access the IP group

**DESCRIPTION** This library provides MIB-II services for the IP group. It provides routines to initialize the group, access the group scalar variables, read the table IP address, route and ARP table. The route and ARP table can also be modified. For a broader description of MIB-II services, see the manual entry for **m2Lib**.

To use this feature, include the following component: **INCLUDE\_MIB2\_IP**

### USING THIS LIBRARY

To use this library, the MIB-II interface group must also be initialized; see the manual entry for **m2IfLib**. This library (**m2IpLib**) can be initialized and deleted by calling **m2IpInit()** and **m2IpDelete()** respectively, if only the IP group's services are needed. If full MIB-II support is used, this group and all other groups can be initialized and deleted by calling **m2Init()** and **m2Delete()**.

The following example demonstrates how to access and change IP scalar variables:

```
M2_IP  ipVars;
int    varToSet;
if (m2IpGroupInfoGet (&ipVars) == OK)
    /* values in ipVars are valid */
/* if IP is forwarding packets (MIB-II value is 1) turn it off */
if (ipVars.ipForwarding == M2_ipForwarding_forwarding)
    {
        /* Not forwarding (MIB-II value is 2) */
        ipVars.ipForwarding = M2_ipForwarding_not_forwarding;
        varToSet |= M2_IPFORWARDING;
    }
/* change the IP default time to live parameter */
ipVars.ipDefaultTTL = 55;
if (m2IpGroupInfoSet (varToSet, &ipVars) == OK)
    /* values in ipVars are valid */
```

The IP address table is a read-only table. Entries to this table can be retrieved as follows:

```

M2_IPADDR_TBL ipAddrEntry;
/* Specify the index as zero to get the first entry in the table */
ipAddrEntry.ipAdEntAddr = 0; /* Local IP address in host byte order */
/* get the first entry in the table */
if ((m2IpAddrTblEntryGet (M2_NEXT_VALUE, &ipAddrEntry) == OK)
    /* values in ipAddrEntry in the first entry are valid */
/* Process first entry in the table */
/*
 * For the next call, increment the index returned in the previous call.
 * The increment is to the next possible lexicographic entry; for
 * example, if the returned index was 147.11.46.8 the index passed in the
 * next invocation should be 147.11.46.9. If an entry in the table
 * matches the specified index, then that entry is returned.
 * Otherwise the closest entry following it, in lexicographic order,
 * is returned.
 */
/* get the second entry in the table */
if ((m2IpAddrTblEntryGet (M2_NEXT_VALUE, &ipAddrEntryEntry) == OK)
    /* values in ipAddrEntry in the second entry are valid */

```

The IP Address Translation Table (ARP table) includes the functionality of the AT group plus additional functionality. The AT group is supported through this MIB-II table. Entries in this table can be added and deleted. An entry is deleted (with a set operation) by setting the **ipNetToMediaType** field to the MIB-II “invalid” value (2). The following example shows how to delete an entry:

```

M2_IPATRANSTBL atEntry;
/* Specify the index for the connection to be deleted in the table */
atEntry.ipNetToMediaIfIndex = 1 /* interface index */
/* destination IP address in host byte order */
atEntry.ipNetToMediaNetAddress = 0x930b2e08;
/* mark entry as invalid */
atEntry.ipNetToMediaType = M2_ipNetToMediaType_invalid;
/* set the entry in the table */
if ((m2IpAtransTblEntrySet (&atEntry) == OK)
    /* Entry deleted successfully */

```

The IP route table allows for entries to be read, deleted, and modified. This example demonstrates how an existing route is deleted:

```

M2_IPROUTETBL routeEntry;
/* Specify the index for the connection to be deleted in the table */
/* destination IP address in host byte order */
routeEntry.ipRouteDest = 0x930b2e08;
/* mark entry as invalid */
routeEntry.ipRouteType = M2_ipRouteType_invalid;

```

**m2Lib**

```

    /* set the entry in the table */
    if ((m2IpRouteTblEntrySet (M2_IP_ROUTE_TYPE, &routeEntry) == OK)
        /* Entry deleted successfully */

```

**INCLUDE FILES**    m2Lib.h

**SEE ALSO**        m2Lib, m2SysLib, m2IfLib, m2IcmpLib, m2UdpLib, m2TcpLib

---

## m2Lib

**NAME**            m2Lib – MIB-II API library for SNMP agents

**ROUTINES**        m2Init() - initialize the SNMP MIB-2 library  
 m2Delete() - delete all the MIB-II library groups

**DESCRIPTION**    This library provides Management Information Base (MIB-II, defined in RFC 1213) services for applications wishing to have access to MIB parameters.

To use this feature, include the following component: **INCLUDE\_MIB2\_ALL**

There are no specific provisions for MIB-I: all services are provided at the MIB-II level. Applications that use this library for MIB-I must hide the MIB-II extensions from higher level protocols. The library accesses all the MIB-II parameters, and presents them to the application in data structures based on the MIB-II specifications.

The routines provided by the VxWorks MIB-II library are separated into groups that follow the MIB-II definition. Each supported group has its own interface library:

**m2SysLib**  
 systems group

**m2IfLib**  
 interface group

**m2IpLib**  
 IP group (includes AT)

**m2IcmpLib**  
 ICMP group

**m2TcpLib**  
 TCP group

**m2UdpLib**  
 UDP group

MIB-II retains the AT group for backward compatibility, but includes its functionality in the IP group. The EGP and SNMP groups are not supported by this interface. The variables in each group have been subdivided into two types: table entries and scalar variables. Each type has a pair of routines that get and set the variables.

#### USING THIS LIBRARY

There are four types of operations on each group:

- initializing the group
- getting variables and table entries
- setting variables and table entries
- deleting the group

Only the groups that are to be used need be initialized. There is one exception: to use the IP group, the interface group must also be initialized. Applications that require MIB-II support from all groups can initialize all groups at once by calling the **m2Init()**. All MIB-II group services can be disabled by calling **m2Delete()**. Applications that need access only to a particular set of groups need only call the initialization routines of the desired groups.

To read the scalar variables for each group, call one of the following routines:

```
m2SysGroupInfoGet()  
m2IfGroupInfoGet()  
m2IpGroupInfoGet()  
m2IcmpGroupInfoGet()  
m2TcpGroupInfoGet()  
m2UdpGroupInfoGet()
```

The input parameter to the routine is always a pointer to a structure specific to the associated group. The scalar group structures follow the naming convention "M2\_groupname". The get routines fill in the input structure with the values of all the group variables.

The scalar variables can also be set to a user supplied value. Not all groups permit setting variables, as specified by the MIB-II definition. The following group routines allow setting variables:

```
m2SysGroupInfoSet()  
m2IpGroupInfoSet()
```

The input parameters to the variable-set routines are a bit field that specifies which variables to set, and a group structure. The structure is the same structure type used in the get operation. Applications need set only the structure fields corresponding to the bits that are set in the bit field.

The MIB-II table routines read one entry at a time. Each MIB-II group that has tables has a get routine for each table. The following table-get routines are available:

```
m2IfTblEntryGet()  
m2IpAddrTblEntryGet()
```

**m2IpAtransTblEntryGet()**  
**m2IpRouteTblEntryGet()**  
**m2TcpConnEntryGet()**  
**m2UdpTblEntryGet()**

The input parameters are a pointer to a table entry structure, and a flag value specifying one of two types of table search. Each table entry is a structure, where the struct type name follows this naming convention: "M2\_GroupnameTablenameTBL". The MIB-II RFC specifies an index that identifies a table entry. Each get request must specify an index value. To retrieve the first entry in a table, set all the index fields of the table-entry structure to zero, and use the search parameter M2\_NEXT\_VALUE. To retrieve subsequent entries, pass the index returned from the previous invocation, incremented to the next possible lexicographical entry. The search field can only be set to the constants M2\_NEXT\_VALUE or M2\_EXACT\_VALUE:

**M2\_NEXT\_VALUE**

retrieves a table entry that is either identical to the index value specified as input, or is the closest entry following that value, in lexicographic order.

**M2\_EXACT\_VALUE**

retrieves a table entry that exactly matches the index specified in the input structure.

Some MIB-II table entries can be added, modified and deleted. Routines to manipulate such entries are described in the manual pages for individual groups.

All the IP network addresses that are exchanged with the MIB-II library must be in host-byte order; use **ntohl()** to convert addresses before calling these library routines.

The following example shows how to initialize the MIB-II library for all groups.

```
extern FUNCPTR myTrapGenerator;
extern void * myTrapGeneratorArg;
M2_OBJECTID mySysObjectId = { 8, {1,3,6,1,4,1,731,1} };
if (m2Init ("VxWorks 5.1.1 MIB-II library (sysDescr)",
           "support@wrs.com (sysContact)",
           "500 Wind River Way Alameda, California 94501 (sysLocation)",
           &mySysObjectId,
           myTrapGenerator,
           myTrapGeneratorArg,
           0) == OK)
    /* MIB-II groups initialized successfully */
```

**INCLUDE FILES**    m2Lib.h

**SEE ALSO**        m2IfLib, m2IpLib, m2IcmpLib, m2UdpLib, m2TcpLib, m2SysLib

---

## m2RipLib

<b>NAME</b>	<b>m2RipLib</b> – VxWorks interface routines to RIP for SNMP Agent
<b>ROUTINES</b>	<b>m2RipInit()</b> - initialize the RIP MIB support <b>m2RipDelete()</b> - delete the RIP MIB support <b>m2RipGlobalCountersGet()</b> - get MIB-II RIP-group global counters <b>m2RipIfStatEntryGet()</b> - get MIB-II RIP-group interface entry <b>m2RipIfConfEntryGet()</b> - get MIB-II RIP-group interface entry <b>m2RipIfConfEntrySet()</b> - set MIB-II RIP-group interface entry
<b>DESCRIPTION</b>	This library provides routines to initialize the group, access the group global variables, read the table of network interfaces that RIP knows about, and change the state of such an interface. For a broader description of MIB-II services, see the manual entry for <b>m2Lib</b> .

### USING THIS LIBRARY

This library can be initialized and deleted by calling **m2RipInit()** and **m2RipDelete()** respectively, if only the RIP group's services are needed. If full MIB-II support is used, this group and all other groups can be initialized and deleted by calling **m2Init()** and **m2Delete()**.

The group global variables are accessed by calling **m2RipGlobalCountersGet()** as follows:

```
M2_RIP2_GLOBAL_GROUP  ripGlobal;
if (m2RipGlobalCountersGet (&ripGlobal) == OK)
    /* values in ripGlobal are valid */
```

To retrieve the RIP group statistics for a particular interface you call the **m2RipIfStatEntryGet()** routine a pointer to an **M2\_RIP2\_IFSTAT\_ENTRY** structure that contains the address of the interface you are searching for. For example:

```
M2_RIP2_IFSTAT_ENTRY ripIfStat;

ripIfStat.rip2IfStatAddress = inet_addr("90.0.0.3");
if (m2RipIfStatEntryGet(M2_EXACT_VALUE, &ripIfStat) == OK)
    /* values in ripIfState are valid */
```

To retrieve the configuration statistics for a particular interface the **m2RipIfConfEntryGet()** routine must be called with an IP address encoded in an **M2\_RIP2\_IFCONF\_ENTRY** structure which is passed as the second argument. For example:

```
M2_RIP2_IFCONF_ENTRY ripIfConf;

ripIfConf.rip2IfConfAddress = inet_addr("90.0.0.3");
if (m2RipIfConfEntryGet(M2_EXACT_VALUE, &ripIfConf) == OK)
    /* values in ripIfConf are valid */
```

To set the values of for an interface the **m2RipIfConfEntrySet()** routine must be called with an IP address in dot notation encoded into an **M2\_RIP2\_IFSTAT\_ENTRY** structure, which is passed as the second argument. For example:

```
M2_RIP2_IFCONF_ENTRY ripIfConf;
    ripIfConf.rip2IfConfAddress = inet_addr("90.0.0.3");
    /* Set the authorization type. */
    ripIfConf.rip2IfConfAuthType = M2_rip2IfConfAuthType_simplePassword;
    bzero(ripIfConf.rip2IfConfAuthKey, 16);
    bcopy("Simple Password ", ripIfConf.rip2IfConfAuthKey, 16);
    /* We only accept version 1 packets. */
    ripIfConf.rip2IfConfSend = M2_rip2IfConfSend_ripVersion1;
    /* We only send version 1 packets. */
    ripIfConf.rip2IfConfReceive = M2_rip2IfConfReceive_rip1;
    /* Default routes have a metric of 2 */
    ripIfConf.rip2IfConfDefaultMetric = 2;
    /* If the interface is invalid it is turned off, we make it valid. */
    ripIfConf.rip2IfConfStatus = M2_rip2IfConfStatus_valid;

    if (m2RipIfConfEntrySet(varsToSet, &ripIfConf) == OK)
        /* Call succeeded. */
```

**INCLUDE FILES**    rip/m2RipLib.h, rip/defs.h

**SEE ALSO**        ripLib

---

## m2SysLib

**NAME**            m2SysLib – MIB-II system-group API for SNMP agents

**ROUTINES**        m2SysInit() - initialize MIB-II system-group routines  
                  m2SysGroupInfoGet() - get system-group MIB-II variables  
                  m2SysGroupInfoSet() - set system-group MIB-II variables to new values  
                  m2SysDelete() - delete resources used to access the MIB-II system group

**DESCRIPTION**    This library provides MIB-II services for the system group. It provides routines to initialize the group and to access the group scalar variables. For a broader description of MIB-II services, see the manual entry for **m2Lib**.

To use this feature, include the following component: **INCLUDE\_MIB2\_SYSTEM**

**USING THIS LIBRARY**

This library can be initialized and deleted by calling **m2SysInit()** and **m2SysDelete()**

respectively, if only the system group's services are needed. If full MIB-II support is used, this group and all other groups can be initialized and deleted by calling `m2Init()` and `m2Delete()`.

The system group provides the option to set the system variables at the time `m2Sysinit()` is called. The MIB-II variables `sysDescr` and `sysobjectId` are read-only, and can be set only by the system-group initialization routine. The variables `sysContact`, `sysName` and `sysLocation` can be set through `m2SysGroupInfoSet()` at any time.

The following is an example of system group initialization:

```
M2_OBJECTID mySysObjectId = { 8, {1,3,6,1,4,1,731,1} };
if (m2SysInit ("VxWorks MIB-II library ",
              "support@wrs.com",
              "1010 Atlantic Avenue Alameda, California 94501",
              &mySysObjectId) == OK)
    /* System group initialized successfully */
```

The system group variables can be accessed as follows:

```
M2_SYSTEM sysVars;
if (m2SysGroupInfoGet (&sysVars) == OK)
    /* values in sysVars are valid */
```

The system group variables can be set as follows:

```
M2_SYSTEM sysVars;
unsigned int varToSet; /* bit field of variables to set */
/* Set the new system Name */
strcpy (m2SysVars.sysName, "New System Name");
varToSet |= M2SYSNAME;
/* Set the new contact name */
strcpy (m2SysVars.sysContact, "New Contact");
varToSet |= M2SYSCONTACT;
if (m2SysGroupInfoGet (varToSet, &sysVars) == OK)
    /* values in sysVars set */
```

INCLUDE FILES    `m2Lib.h`

SEE ALSO        `m2Lib`, `m2IfLib`, `m2IpLib`, `m2IcmpLib`, `m2UdpLib`, `m2TcpLib`

---

## m2TcpLib

<b>NAME</b>	<b>m2TcpLib</b> – MIB-II TCP-group API for SNMP agents
<b>ROUTINES</b>	<b>m2TcpInit()</b> - initialize MIB-II TCP-group access <b>m2TcpGroupInfoGet()</b> - get MIB-II TCP-group scalar variables <b>m2TcpConnEntryGet()</b> - get a MIB-II TCP connection table entry <b>m2TcpConnEntrySet()</b> - set a TCP connection to the closed state <b>m2TcpDelete()</b> - delete all resources used to access the TCP group
<b>DESCRIPTION</b>	This library provides MIB-II services for the TCP group. It provides routines to initialize the group, access the group global variables, read the table of TCP connections, and change the state of a TCP connection. For a broader description of MIB-II services, see the manual entry for <b>m2Lib</b> .  To use this feature, include the following component: <b>INCLUDE_MIB2_TCP</b>

### USING THIS LIBRARY

This library can be initialized and deleted by calling **m2TcpInit()** and **m2TcpDelete()** respectively, if only the TCP group's services are needed. If full MIB-II support is used, this group and all other groups can be initialized and deleted by calling **m2Init()** and **m2Delete()**.

The group global variables are accessed by calling **m2TcpGroupInfoGet()** as follows:

```
M2_TCP    tcpVars;  
if (m2TcpGroupInfoGet (&tcpVars) == OK)  
    /* values in tcpVars are valid */
```

The TCP table of connections can be accessed in lexicographical order. The first entry in the table can be accessed by setting the table index to zero. Every other entry thereafter can be accessed by passing to **m2TcpConnTblEntryGet()** the index retrieved in the previous invocation incremented to the next lexicographical value by giving **M2\_NEXT\_VALUE** as the search parameter. For example:

```
M2_TCPCONNTBL  tcpEntry;  
/* Specify a zero index to get the first entry in the table */  
tcpEntry.tcpConnLocalAddress = 0; /* Lcl IP address in host byte order */  
tcpEntry.tcpConnLocalPort    = 0; /* Local TCP port                      */  
tcpEntry.tcpConnRemAddress   = 0; /* remote IP address                  */  
tcpEntry.tcpConnRemPort     = 0; /* remote TCP port in host byte order */  
/* get the first entry in the table */  
if ((m2TcpConnTblEntryGet (M2_NEXT_VALUE, &tcpEntry) == OK)  
    /* values in tcpEntry in the first entry are valid */  
    /* process first entry in the table */  
    /*
```

```

* For the next call, increment the index returned in the previous call.
* The increment is to the next possible lexicographic entry; for
* example, if the returned index was 147.11.46.8.2000.147.11.46.158.1000
* the index passed in the next invocation should be
* 147.11.46.8.2000.147.11.46.158.1001. If an entry in the table
* matches the specified index, then that entry is returned.
* Otherwise the closest entry following it, in lexicographic order,
* is returned.
*/
/* get the second entry in the table */
if ((m2TcpConnTblEntryGet (M2_NEXT_VALUE, &tcpEntry) == OK)
    /* values in tcpEntry in the second entry are valid */

```

The TCP table of connections allows only for a connection to be deleted as specified in the MIB-II. For example:

```

M2_TCPCONNTBL tcpEntry;
/* Fill in the index for the connection to be deleted in the table */
/* Local IP address in host byte order, and local port number */
tcpEntry.tcpConnLocalAddress = 0x930b2e08;
tcpEntry.tcpConnLocalPort    = 3000;
/* Remote IP address in host byte order, and remote port number */
tcpEntry.tcpConnRemAddress   = 0x930b2e9e;
tcpEntry.tcpConnRemPort      = 3000;
tcpEntry.tcpConnState        = 12; /* MIB-II state value for delete */
/* set the entry in the table */
if ((m2TcpConnTblEntrySet (&tcpEntry) == OK)
    /* tcpEntry deleted successfully */

```

INCLUDE FILES    m2Lib.h

SEE ALSO        m2Lib, m2IfLib, m2IpLib, m2IcmpLib, m2UdpLib, m2SysLib

M

---

## m2UdpLib

<b>NAME</b>	<b>m2UdpLib</b> – MIB-II UDP-group API for SNMP agents
<b>ROUTINES</b>	<b>m2UdpInit()</b> - initialize MIB-II UDP-group access <b>m2UdpGroupInfoGet()</b> - get MIB-II UDP-group scalar variables <b>m2UdpTblEntryGet()</b> - get a UDP MIB-II entry from the UDP list of listeners <b>m2UdpDelete()</b> - delete all resources used to access the UDP group
<b>DESCRIPTION</b>	This library provides MIB-II services for the UDP group. It provides routines to initialize the group, access the group scalar variables, and read the table of UDP listeners. For a broader description of MIB-II services, see the manual entry for <b>m2Lib</b> .

To use this feature, include the following component: **INCLUDE\_MIB2\_UDP**

### USING THIS LIBRARY

This library can be initialized and deleted by calling **m2UdpInit()** and **m2UdpDelete()** respectively, if only the UDP group's services are needed. If full MIB-II support is used, this group and all other groups can be initialized and deleted by calling **m2Init()** and **m2Delete()**.

The group scalar variables are accessed by calling **m2UdpGroupInfoGet()** as follows:

```
M2_UDP   udpVars;  
if (m2UdpGroupInfoGet (&udpVars) == OK)  
    /* values in udpVars are valid */
```

The UDP table of listeners can be accessed in lexicographical order. The first entry in the table can be accessed by setting the table index to zero in a call to **m2UdpTblEntryGet()**. Every other entry thereafter can be accessed by incrementing the index returned from the previous invocation to the next possible lexicographical index, and repeatedly calling **m2UdpTblEntryGet()** with the **M2\_NEXT\_VALUE** constant as the search parameter. For example:

```
M2_UDPTBL  udpEntry;  
/* Specify zero index to get the first entry in the table */  
udpEntry.udpLocalAddress = 0; /* local IP Address in host byte order */  
udpEntry.udpLocalPort    = 0; /* local port Number */  
/* get the first entry in the table */  
if ((m2UdpTblEntryGet (M2_NEXT_VALUE, &udpEntry) == OK)  
    /* values in udpEntry in the first entry are valid */  
/* process first entry in the table */  
/*  
 * For the next call, increment the index returned in the previous call.  
 * The increment is to the next possible lexicographic entry; for  
 * example, if the returned index was 0.0.0.0.3000 the index passed in
```

```

* the next invocation should be 0.0.0.0.3001.  If an entry in the table
* matches the specified index, then that entry is returned.
* Otherwise the closest entry following it, in lexicographic order,
* is returned.
*/
/* get the second entry in the table */
if ((m2UdpTblEntryGet (M2_NEXT_VALUE, &udpEntry) == OK)
    /* values in udpEntry in the second entry are valid */

```

INCLUDE FILES    **m2Lib.h**

SEE ALSO        **m2Lib, m2IfLib, m2IpLib, m2IcmpLib, m2TcpLib, m2SysLib**

---

## mathALib

NAME            **mathALib** – C interface library to high-level math functions

ROUTINES

- acos()** - compute an arc cosine (ANSI)
- asin()** - compute an arc sine (ANSI)
- atan()** - compute an arc tangent (ANSI)
- atan2()** - compute the arc tangent of y/x (ANSI)
- cbrt()** - compute a cube root
- ceil()** - compute the smallest integer greater than or equal to a specified value (ANSI)
- cos()** - compute a cosine (ANSI)
- cosh()** - compute a hyperbolic cosine (ANSI)
- exp()** - compute an exponential value (ANSI)
- fabs()** - compute an absolute value (ANSI)
- floor()** - compute the largest integer less than or equal to a specified value (ANSI)
- fmod()** - compute the remainder of x/y (ANSI)
- infinity()** - return a very large double
- rint()** - convert a double-precision value to an integer
- iround()** - round a number to the nearest integer
- log()** - compute a natural logarithm (ANSI)
- log10()** - compute a base-10 logarithm (ANSI)
- log2()** - compute a base-2 logarithm
- pow()** - compute the value of a number raised to a specified power (ANSI)
- round()** - round a number to the nearest integer
- sin()** - compute a sine (ANSI)
- sincos()** - compute both a sine and cosine
- sinh()** - compute a hyperbolic sine (ANSI)
- sqrt()** - compute a non-negative square root (ANSI)
- tan()** - compute a tangent (ANSI)

**tanh()** - compute a hyperbolic tangent (ANSI)  
**trunc()** - truncate to integer  
**acosf()** - compute an arc cosine (ANSI)  
**asinf()** - compute an arc sine (ANSI)  
**atanf()** - compute an arc tangent (ANSI)  
**atan2f()** - compute the arc tangent of  $y/x$  (ANSI)  
**cbrtf()** - compute a cube root  
**ceilf()** - compute the smallest integer greater than or equal to a specified value (ANSI)  
**cosf()** - compute a cosine (ANSI)  
**coshf()** - compute a hyperbolic cosine (ANSI)  
**expf()** - compute an exponential value (ANSI)  
**fabsf()** - compute an absolute value (ANSI)  
**floorf()** - compute the largest integer less than or equal to a specified value (ANSI)  
**fmodf()** - compute the remainder of  $x/y$  (ANSI)  
**infinityf()** - return a very large float  
**irintf()** - convert a single-precision value to an integer  
**roundf()** - round a number to the nearest integer  
**logf()** - compute a natural logarithm (ANSI)  
**log10f()** - compute a base-10 logarithm (ANSI)  
**log2f()** - compute a base-2 logarithm  
**powf()** - compute the value of a number raised to a specified power (ANSI)  
**roundf()** - round a number to the nearest integer  
**sinf()** - compute a sine (ANSI)  
**sincosf()** - compute both a sine and cosine  
**sinhf()** - compute a hyperbolic sine (ANSI)  
**sqrtf()** - compute a non-negative square root (ANSI)  
**tanf()** - compute a tangent (ANSI)  
**tanhf()** - compute a hyperbolic tangent (ANSI)  
**truncf()** - truncate to integer

**DESCRIPTION** This library provides a C interface to high-level floating-point math functions, which can use either a hardware floating-point unit or a software floating-point emulation library. The appropriate routine is called based on whether **mathHardInit()** or **mathSoftInit()** or both have been called to initialize the interface.

All angle-related parameters are expressed in radians. All functions in this library with names corresponding to ANSI C specifications are ANSI compatible.

---

**WARNING:** Not all functions in this library are available on all architectures. For information on available math functions, consult the VxWorks architecture supplement for your processor.

---

**INCLUDE FILES** **math.h**

**SEE ALSO** **ansiMath**, **fppLib**, **floatLib**, **mathHardLib**, **mathSoftLib**, the various *Architecture Supplements*, Kernighan & Ritchie: *The C Programming Language, 2nd Edition*

---

## mathHardLib

<b>NAME</b>	<b>mathHardLib</b> – hardware floating-point math library
<b>ROUTINES</b>	<b>mathHardInit()</b> - initialize hardware floating-point math support
<b>DESCRIPTION</b>	<p>This library provides support routines for using hardware floating-point units with high-level math functions. The high-level functions include trigonometric operations, exponents, and so forth.</p> <p>The routines in this library are used automatically for high-level math functions only if <b>mathHardInit()</b> has been called previously.</p> <hr/> <p><b>WARNING:</b> Not all architectures support hardware floating-point. See the architecture-specific appendices of the <i>VxWorks Programmer's Guide</i>.</p> <hr/>
<b>INCLUDE FILES</b>	<b>math.h</b>
<b>SEE ALSO</b>	<b>mathSoftLib</b> , <b>mathALib</b> , <i>VxWorks Programmer's Guide</i> architecture-specific appendices

---

## mathSoftLib

<b>NAME</b>	<b>mathSoftLib</b> – high-level floating-point emulation library
<b>ROUTINES</b>	<b>mathSoftInit()</b> - initialize software floating-point math support
<b>DESCRIPTION</b>	<p>This library provides software emulation of various high-level floating-point operations. This emulation is generally for use in systems that lack a floating-point coprocessor.</p> <hr/> <p><b>WARNING:</b> Software floating point is not supported for all architectures. See the architecture-specific appendices of the <i>VxWorks Programmer's Guide</i>.</p> <hr/>
<b>INCLUDE FILES</b>	<b>math.h</b>
<b>SEE ALSO</b>	<b>mathHardLib</b> , <b>mathALib</b> , <i>VxWorks Programmer's Guide</i> architecture-specific appendices

## memDrv

<b>NAME</b>	<b>memDrv</b> – pseudo-memory device driver
<b>ROUTINES</b>	<b>memDrv()</b> - install a memory driver <b>memDevCreate()</b> - create a memory device <b>memDevCreateDir()</b> - create a memory device for multiple files <b>memDevDelete()</b> - delete a memory device
<b>DESCRIPTION</b>	<p>This driver allows the I/O system to access memory directly as a pseudo-I/O device. Memory location and size are specified when the device is created. This feature is useful when data must be preserved between boots of VxWorks or when sharing data between CPUs.</p> <p>Additionally, it can be used to build some files into a VxWorks binary image (having first converted them to data arrays in C source files, using a utility such as <b>memdrvbuild</b>), and then mount them in the file system; this is a simple way of delivering some non-changing files with VxWorks. For example, a system with an integrated web server may use this technique to build some HTML and associated content files into VxWorks.</p> <p><b>memDrv</b> can be used to simply provide a high-level method of reading and writing bytes in absolute memory locations through I/O calls. It can also be used to implement a simple, essentially read-only file system (existing files can be rewritten within their existing sizes); directory searches and a limited set of IOCTL calls (including <b>stat()</b>) are supported.</p>
<b>USER-CALLABLE ROUTINES</b>	<p>Most of the routines in this driver are accessible only through the I/O system. Four routines, however, can be called directly: <b>memDrv()</b> to initialize the driver, <b>memDevCreate()</b> and <b>memDevCreateDir()</b> to create devices, and <b>memDevDelete()</b> to delete devices.</p> <p>Before using the driver, it must be initialized by calling <b>memDrv()</b>. This routine should be called only once, before any reads, writes, or <b>memDevCreate()</b> calls. It may be called from <b>usrRoot()</b> in <b>usrConfig.c</b> or at some later point.</p>
<b>IOCTL FUNCTIONS</b>	<p>The dosFs file system supports the following <b>ioctl()</b> functions. The functions listed are defined in the header <b>ioLib.h</b>. Unless stated otherwise, the file descriptor used for these functions may be any file descriptor which is opened to a file or directory on the volume or to the volume itself.</p>

### FIOGETFL

Copies to *flags* the open mode flags of the file (O\_RDONLY, O\_WRONLY, O\_RDWR):

```
int flags;  
status = ioctl (fd, FIOGETFL, &flags);
```

**FIOSEEK**

Sets the current byte offset in the file to the position specified by *newOffset*:

```
status = ioctl (fd, FIOSEEK, newOffset);
```

The FIOSEEK offset is always relative to the beginning of the file. The offset, if any, given at open time by using pseudo-file name is overridden.

**FIOWHERE**

Returns the current byte position in the file. This is the byte offset of the next byte to be read or written. It takes no additional argument:

```
position = ioctl (fd, FIOWHERE, 0);
```

**FIONREAD**

Copies to *unreadCount* the number of unread bytes in the file:

```
int unreadCount;  
status = ioctl (fd, FIONREAD, &unreadCount);
```

**FIOREADDIR**

Reads the next directory entry. The argument *dirStruct* is a DIR directory descriptor. Normally, the **readdir()** routine is used to read a directory, rather than using the FIOREADDIR function directly. See **dirLib**.

```
DIR dirStruct;  
fd = open ("directory", O_RDONLY);  
status = ioctl (fd, FIOREADDIR, &dirStruct);
```

**FIOFSTATGET**

Gets file status information (directory entry data). The argument *statStruct* is a pointer to a stat structure that is filled with data describing the specified file. File inode numbers, user and group IDs, and times are not supported (returned as 0).

Normally, the **stat()** or **fstat()** routine is used to obtain file information, rather than using the FIOFSTATGET function directly. See **dirLib**.

```
struct stat statStruct;  
fd = open ("file", O_RDONLY);  
status = ioctl (fd, FIOFSTATGET, &statStruct);
```

Any other **ioctl()** function codes will return error status.

**SEE ALSO**

*VxWorks Programmer's Guide: I/O System*

---

## memLib

<b>NAME</b>	<b>memLib</b> – full-featured memory partition manager
<b>ROUTINES</b>	<b>memPartOptionsSet()</b> - set the debug options for a memory partition <b>memalign()</b> - allocate aligned memory <b>valloc()</b> - allocate memory on a page boundary <b>memPartRealloc()</b> - reallocate a block of memory in a specified partition <b>memPartFindMax()</b> - find the size of the largest available free block <b>memOptionsSet()</b> - set the debug options for the system memory partition <b>calloc()</b> - allocate space for an array (ANSI) <b>realloc()</b> - reallocate a block of memory (ANSI) <b>cfree()</b> - free a block of memory <b>memFindMax()</b> - find the largest free block in the system memory partition
<b>DESCRIPTION</b>	<p>This library provides full-featured facilities for managing the allocation of blocks of memory from ranges of memory called memory partitions. The library is an extension of <b>memPartLib</b> and provides enhanced memory management features, including error handling, aligned allocation, and ANSI allocation routines. For more information about the core memory partition management facility, see the manual entry for <b>memPartLib</b>.</p> <p>The system memory partition is created when the kernel is initialized by <b>kernelInit()</b>, which is called by the root task, <b>usrRoot()</b>, in <b>usrConfig.c</b>. The ID of the system memory partition is stored in the global variable <b>memSysPartId</b>; its declaration is included in <b>memLib.h</b>.</p> <p>The <b>memalign()</b> routine is provided for allocating memory aligned to a specified boundary.</p> <p>This library includes three ANSI-compatible routines: <b>calloc()</b> allocates a block of memory for an array; <b>realloc()</b> changes the size of a specified block of memory; and <b>cfree()</b> returns to the free memory pool a block of memory that was previously allocated with <b>calloc()</b>.</p>
<b>ERROR OPTIONS</b>	<p>Various debug options can be selected for each partition using <b>memPartOptionsSet()</b> and <b>memOptionsSet()</b>. Two kinds of errors are detected: attempts to allocate more memory than is available, and bad blocks found when memory is freed. In both cases, the error status is returned. There are four error-handling options that can be individually selected:</p> <p><b>MEM_ALLOC_ERROR_LOG_FLAG</b> Log a message when there is an error in allocating memory.</p> <p><b>MEM_ALLOC_ERROR_SUSPEND_FLAG</b> Suspend the task when there is an error in allocating memory (unless the task was spawned with the <b>VX_UNBREAKABLE</b> option, in which case it cannot be suspended).</p>

**MEM\_BLOCK\_ERROR\_LOG\_FLAG**

Log a message when there is an error in freeing memory.

**MEM\_BLOCK\_ERROR\_SUSPEND\_FLAG**

Suspend the task when there is an error in freeing memory (unless the task was spawned with the **VX\_UNBREAKABLE** option, in which case it cannot be suspended).

When the following option is specified to check every block freed to the partition, **memPartFree()** and **free()** in **memPartLib** run consistency checks of various pointers and values in the header of the block being freed. If this flag is not specified, no check will be performed when memory is freed.

**MEM\_BLOCK\_CHECK**

Check each block freed.

Setting either of the **MEM\_BLOCK\_ERROR** options automatically sets **MEM\_BLOCK\_CHECK**.

The default options when a partition is created are:

```
MEM_ALLOC_ERROR_LOG_FLAG
MEM_BLOCK_CHECK
MEM_BLOCK_ERROR_LOG_FLAG
MEM_BLOCK_ERROR_SUSPEND_FLAG
```

When setting options for a partition with **memPartOptionsSet()** or **memOptionsSet()**, use the logical OR operator between each specified option to construct the *options* parameter. For example:

```
memPartOptionsSet (myPartId, MEM_ALLOC_ERROR_LOG_FLAG |
MEM_BLOCK_CHECK |
MEM_BLOCK_ERROR_LOG_FLAG);
```

**INCLUDE FILES**     **memLib.h**

**SEE ALSO**         **memPartLib, smMemLib**

**M**

---

## memPartLib

**NAME** **memPartLib** – core memory partition manager

**ROUTINES** **memPartCreate()** - create a memory partition  
**memPartAddToPool()** - add memory to a memory partition  
**memPartAlignedAlloc()** - allocate aligned memory from a partition  
**memPartAlloc()** - allocate a block of memory from a partition  
**memPartFree()** - free a block of memory in a partition  
**memAddToPool()** - add memory to the system memory partition  
**malloc()** - allocate a block of memory from the system memory partition (ANSI)  
**free()** - free a block of memory (ANSI)

**DESCRIPTION** This library provides core facilities for managing the allocation of blocks of memory from ranges of memory called memory partitions. The library was designed to provide a compact implementation; full-featured functionality is available with **memLib**, which provides enhanced memory management features built as an extension of **memPartLib**. (For more information about enhanced memory partition management options, see the manual entry for **memLib**.) This library consists of two sets of routines. The first set, **memPart...()**, comprises a general facility for the creation and management of memory partitions, and for the allocation and deallocation of blocks from those partitions. The second set provides a traditional ANSI-compatible **malloc()/free()** interface to the system memory partition.

The system memory partition is created when the kernel is initialized by **kernelInit()**, which is called by the root task, **usrRoot()**, in **usrConfig.c**. The ID of the system memory partition is stored in the global variable **memSysPartId**; its declaration is included in **memLib.h**.

The allocation of memory, using **malloc()** in the typical case and **memPartAlloc()** for a specific memory partition, is done with a first-fit algorithm. Adjacent blocks of memory are coalesced when they are freed with **memPartFree()** and **free()**. There is also a routine provided for allocating memory aligned to a specified boundary from a specific memory partition, **memPartAlignedAlloc()**.

**CAVEATS** Architectures have various alignment constraints. To provide optimal performance, **malloc()** returns a pointer to a buffer having the appropriate alignment for the architecture in use. The portion of the allocated buffer reserved for system bookkeeping, known as the overhead, may vary depending on the architecture.

Architecture	Boundary	Overhead
ARM	4	8
COLDFIRE	4	8
I86	4	8

Architecture	Boundary	Overhead
M68K	4	8
MCORE	8	8
MIPS	16	16
PPC *	8/16	8/16
SH	4	8
SimNT	8	8
SimSolaris	8	8

\* On PowerPC, the boundary and overhead values are 16 bytes for system based on the PPC604 CPU type (including ALTIVEC). For all other PowerPC CPU types (PPC403, PPC405, PPC440, PPC860, PPC603, *etc....*), the boundary and overhead are 8 bytes.

**INCLUDE FILES**    **memLib.h, stdlib.h**

**SEE ALSO**        **memLib, smMemLib**

---

## memShow

M

**NAME**            **memShow** – memory show routines

**ROUTINES**      **memShowInit()** - initialize the memory partition show facility  
**memShow()** - show system memory partition blocks and statistics  
**memPartShow()** - show partition blocks and statistics  
**memPartInfoGet()** - get partition information

**DESCRIPTION**    This library contains memory partition information display routines. To use this facility, it must first be installed using **memShowInit()**, which is called automatically when the memory partition show facility is configured into VxWorks using either of the following methods:

If you use the configuration header files, define **INCLUDE\_SHOW\_ROUTINES** in **config.h**.

If you use the Tornado project facility, select **INCLUDE\_MEM\_SHOW**.

**SEE ALSO**        **memLib, memPartLib**, *VxWorks Programmer's Guide: Target Shell*, **windsh**, *Tornado User's Guide: Shell*

## **mmanPxLib**

<b>NAME</b>	<b>mmanPxLib</b> – memory management library (POSIX)
<b>ROUTINES</b>	<b>mlockall()</b> - lock all pages used by a process into memory (POSIX) <b>munlockall()</b> - unlock all pages used by a process (POSIX) <b>mlock()</b> - lock specified pages into memory (POSIX) <b>munlock()</b> - unlock specified pages (POSIX)
<b>DESCRIPTION</b>	This library contains POSIX interfaces designed to lock and unlock memory pages, <i>i.e.</i> , to control whether those pages may be swapped to secondary storage. Since VxWorks does not use swapping (all pages are always kept in memory), these routines have no real effect and simply return 0 ( <b>OK</b> ).
<b>INCLUDE FILES</b>	sys/mman.h
<b>SEE ALSO</b>	POSIX 1003.1b document

---

## **mmuMapLib**

<b>NAME</b>	<b>mmuMapLib</b> – MMU mapping library for ARM Ltd. processors
<b>ROUTINES</b>	<b>mmuVirtToPhys()</b> - translate a virtual address to a physical address (ARM) <b>mmuPhysToVirt()</b> - translate a physical address to a virtual address (ARM)
<b>DESCRIPTION</b>	This library provides additional MMU support routines. These are present in a separate module from <b>mmuLib.c</b> , so that these routines can be used without including all the code in that object module.

---

## mmuPro32Lib

**NAME** **mmuPro32Lib** – MMU library for PentiumPro/2/3/4 32 bit mode

**ROUTINES** **mmuPro32LibInit()** - initialize module

**DESCRIPTION** **mmuPro32Lib.c** provides the architecture dependent routines that directly control the memory management unit. It provides 10 routines that are called by the higher level architecture independent routines in **vmLib.c**:

**mmuPro32LibInit()** - initialize module  
**mmuTransTblCreate()** - create a new translation table  
**mmuTransTblDelete()** - delete a translation table.  
**mmuPro32Enable()** - turn MMU on or off  
**mmuStateSet()** - set state of virtual memory page  
**mmuStateGet()** - get state of virtual memory page  
**mmuPageMap()** - map physical memory page to virtual memory page  
**mmuGlobalPageMap()** - map physical memory page to global virtual memory page  
**mmuTranslate()** - translate a virtual address to a physical address  
**mmuCurrentSet()** - change active translation table

Applications using the MMU will never call these routines directly; the visible interface is supported in **vmLib.c**.

**mmuLib** supports the creation and maintenance of multiple translation tables, one of which is the active translation table when the MMU is enabled. Note that VxWorks does not include a translation table as part of the task context; individual tasks do not reside in private virtual memory. However, we include the facilities to create multiple translation tables so that the user may create “private” virtual memory contexts and switch them in an application specific manner. New translation tables are created with a call to **mmuTransTblCreate()**, and installed as the active translation table with **mmuCurrentSet()**. Translation tables are modified and potentially augmented with calls to **mmuPageMap()** and **mmuStateSet()**. The state of portions of the translation table can be read with calls to **mmuStateGet()** and **mmuTranslate()**.

The traditional VxWorks architecture and design philosophy requires that all objects and operating systems resources be visible and accessible to all agents (tasks, isrs, watchdog timers, *etc.*) in the system. This has traditionally been insured by the fact that all objects and data structures reside in physical memory; thus, a data structure created by one agent may be accessed by any other agent using the same pointer (object identifiers in VxWorks are often pointers to data structures.) This creates a potential problem if you have multiple virtual memory contexts. For example, if a semaphore is created in one virtual memory context, you must guarantee that that semaphore will be visible in all virtual memory contexts if the semaphore is to be accessed at interrupt level, when a virtual memory context other than the one in which it was created may be active. Another example is that

code loaded using the incremental loader from the shell must be accessible in all virtual memory contexts, since code is shared by all agents in the system.

This problem is resolved by maintaining a global “transparent” mapping of virtual to physical memory for all the contiguous segments of physical memory (on board memory, i/o space, sections of vme space, *etc.*) that is shared by all translation tables; all available physical memory appears at the same address in virtual memory in all virtual memory contexts. This technique provides an environment that allows resources that rely on a globally accessible physical address to run without modification in a system with multiple virtual memory contexts.

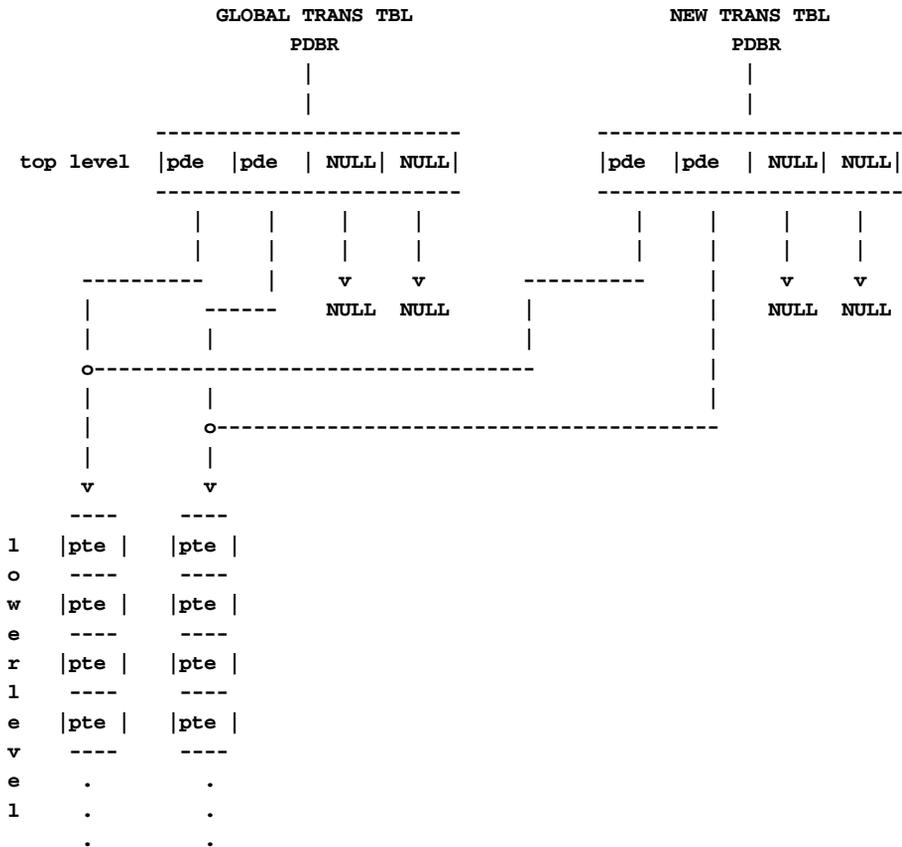
An additional requirement is that modifications made to the state of global virtual memory in one translation table appear in all translation tables. For example, memory containing the text segment is made read only (to avoid accidental corruption) by setting the appropriate writable bits in the translation table entries corresponding to the virtual memory containing the text segment. This state information must be shared by all virtual memory contexts, so that no matter what translation table is active, the text segment is protected from corruption. The mechanism that implements this feature is architecture dependent, but usually entails building a section of a translation table that corresponds to the global memory, that is shared by all other translation tables. Thus, when changes to the state of the global memory are made in one translation table, the changes are reflected in all other translation tables.

**mmuLib** provides a separate call for constructing global virtual memory - **mmuGlobalPageMap()** - which creates translation table entries that are shared by all translation tables. Initialization code in **usrConfig** makes calls to **vmGlobalMap()** (which in turn calls **mmuGlobalPageMap()**) to set up global transparent virtual memory for all available physical memory. All calls made to **mmuGlobalPageMap()** must occur before any virtual memory contexts are created; changes made to global virtual memory after virtual memory contexts are created are not guaranteed to be reflected in all virtual memory contexts.

Most MMU architectures will dedicate some fixed amount of virtual memory to a minimal section of the translation table (a “segment”, or “block”). This creates a problem in that the user may map a small section of virtual memory into the global translation tables, and then attempt to use the virtual memory after this section as private virtual memory. The problem is that the translation table entries for this virtual memory are contained in the global translation tables, and are thus shared by all translation tables. This condition is detected by **vmMap()**, and an error is returned, thus, the lower level routines in **mmuPro32Lib.c** (**mmuPageMap()**, **mmuGlobalPageMap()**) need not perform any error checking.

A global variable **mmuPageBlockSize** should be defined which is equal to the minimum virtual segment size. **mmuLib** must provide a routine **mmuGlobalInfoGet()**, which returns a pointer to the **globalPageBlock[]** array. This provides the user with enough information to be able to allocate virtual memory space that does not conflict with the global memory space.





Note that with this scheme, the global memory granularity is 4MB. Each time you map a section of global virtual memory, you dedicate at least 4MB of the virtual space to global virtual memory that will be shared by all virtual memory contexts.

The physical memory that holds these data structures is obtained from the system memory manager via **memalign()** to ensure that the memory is page aligned. We want to protect this memory from being corrupted, so we invalidate the descriptors that we set up in the global translation that correspond to the memory containing the translation table data structures. This creates a “chicken and the egg” paradox, in that the only way we can modify these data structures is through virtual memory that is now invalidated, and we can’t validate it because the page descriptors for that memory are in invalidated memory (confused yet?) So, you will notice that anywhere that page table descriptors (pte’s) are modified, we do so by locking out interrupts, momentarily disabling the MMU, accessing the memory with its physical address, enabling the MMU, and then re-enabling interrupts (see **mmuStateSet()**, for example.)

Support for two new page attribute bits are added for PentiumPro's enhanced MMU. They are Global bit (G) and Page-level write-through/back bit (PWT). Global bit indicates a global page when set. When a page is marked global and the page global enable (PGE) bit in register CR4 is set, the page-table or page-directory entry for the page is not invalidated in the TLB when register CR3 is loaded or a task switch occurs. This bit is provided to prevent frequently used pages (such as pages that contain kernel or other operating system or executive code) from being flushed from the TLB. Page-level write-through/back bit (PWT) controls the write-through or write-back caching policy of individual pages or page tables. When the PWT bit is set, write-through caching is enabled for the associated page or page table. When the bit is clear, write-back caching is enabled for the associated page and page table. Following macros are used to describe these attribute bits in the physical memory descriptor table `sysPhysMemDesc[]` in `sysLib.c`.

`VM_STATE_WBACK` - use write-back cache policy for the page

`VM_STATE_WBACK_NOT` - use write-through cache policy for the page

`VM_STATE_GLOBAL` - set page global bit

`VM_STATE_GLOBAL_NOT` - not set page global bit

Support for two page size (4KB and 4MB) are added also. The linear address for 4KB pages is divided into three sections:

Page directory entry - bits 22 through 31.

Page table entry - Bits 12 through 21.

Page offset - Bits 0 through 11.

The linear address for 4MB pages is divided into two sections:

Page directory entry - Bits 22 through 31.

Page offset - Bits 0 through 21.

These two page size is configurable by `VM_PAGE_SIZE` macro in `config.h`.

## mmuSh7700Lib

<b>NAME</b>	<b>mmuSh7700Lib</b> – Hitachi SH7700 MMU support library
<b>ROUTINES</b>	<b>mmuSh7700LibInit()</b> - initialize module
<b>DESCRIPTION</b>	<p><b>mmuLib.c</b> provides the architecture dependent routines that directly control the memory management unit. It provides 10 routines that are called by the higher level architecture independent routines in <b>vmLib.c</b>:</p> <ul style="list-style-type: none"><li>– <b>mmuLibInit()</b> - initialize module</li><li>– <b>mmuTransTblCreate()</b> - create a new translation table</li><li>– <b>mmuTransTblDelete()</b> - delete a translation table.</li><li>– <b>mmuEnable()</b> - turn mmu on or off</li><li>– <b>mmuStateSet()</b> - set state of virtual memory page</li><li>– <b>mmuStateGet()</b> - get state of virtual memory page</li><li>– <b>mmuPageMap()</b> - map physical memory page to virtual memory page</li><li>– <b>mmuGlobalPageMap()</b> - map physical memory page to global virtual memory page</li><li>– <b>mmuTranslate()</b> - translate a virtual address to a physical address</li><li>– <b>mmuCurrentSet()</b> - change active translation table</li></ul>

Applications using the mmu will never call these routines directly; the visible interface is supported in **vmLib.c**.

**mmuLib** supports the creation and maintenance of multiple translation tables, one of which is the active translation table when the mmu is enabled. Note that VxWorks does not include a translation table as part of the task context; individual tasks do not reside in private virtual memory. However, we include the facilities to create multiple translation tables so that the user may create “private” virtual memory contexts and switch them in an application specific manner. New translation tables are created with a call to **mmuTransTblCreate()**, and installed as the active translation table with **mmuCurrentSet()**. Translation tables are modified and potentially augmented with calls to **mmuPageMap()** and **mmuStateSet()**. The state of portions of the translation table can be read with calls to **mmuStateGet()** and **mmuTranslate()**.

The traditional VxWorks architecture and design philosophy requires that all objects and operating systems resources be visible and accessible to all agents (tasks, isrs, watchdog timers, *etc.*) in the system. This has traditionally been insured by the fact that all objects and data structures reside in physical memory; thus, a data structure created by one agent may be accessed by any other agent using the same pointer (object identifiers in VxWorks are often pointers to data structures.) This creates a potential problem if you have multiple

virtual memory contexts. For example, if a semaphore is created in one virtual memory context, you must guarantee that that semaphore will be visible in all virtual memory contexts if the semaphore is to be accessed at interrupt level, when a virtual memory context other than the one in which it was created may be active. Another example is that code loaded using the incremental loader from the shell must be accessible in all virtual memory contexts, since code is shared by all agents in the system.

This problem is resolved by maintaining a global “transparent” mapping of virtual to physical memory for all the contiguous segments of physical memory (on board memory, i/o space, sections of vme space, *etc.*) that is shared by all translation tables; all available physical memory appears at the same address in virtual memory in all virtual memory contexts. This technique provides an environment that allows resources that rely on a globally accessible physical address to run without modification in a system with multiple virtual memory contexts.

An additional requirement is that modifications made to the state of global virtual memory in one translation table appear in all translation tables. For example, memory containing the text segment is made read only (to avoid accidental corruption) by setting the appropriate writable bits in the translation table entries corresponding to the virtual memory containing the text segment. This state information must be shared by all virtual memory contexts, so that no matter what translation table is active, the text segment is protected from corruption. The mechanism that implements this feature is architecture dependent, but usually entails building a section of a translation table that corresponds to the global memory, that is shared by all other translation tables. Thus, when changes to the state of the global memory are made in one translation table, the changes are reflected in all other translation tables.

**mmuLib** provides a separate call for constructing global virtual memory - **mmuGlobalPageMap()** - which creates translation table entries that are shared by all translation tables. Initialization code in **usrConfig** makes calls to **vmGlobalMap()** (which in turn calls **mmuGlobalPageMap()**) to set up global transparent virtual memory for all available physical memory. All calls made to **mmuGlobalPageMap()** must occur before any virtual memory contexts are created; changes made to global virtual memory after virtual memory contexts are created are not guaranteed to be reflected in all virtual memory contexts.

Most mmu architectures will dedicate some fixed amount of virtual memory to a minimal section of the translation table (a “segment”, or “block”). This creates a problem in that the user may map a small section of virtual memory into the global translation tables, and then attempt to use the virtual memory after this section as private virtual memory. The problem is that the translation table entries for this virtual memory are contained in the global translation tables, and are thus shared by all translation tables. This condition is detected by **vmMap()**, and an error is returned, thus, the lower level routines in **mmuLib.c** (**mmuPageMap()**, **mmuGlobalPageMap()**) need not perform any error checking.

A global variable called **mmuPageBlockSize** should be defined which is equal to the minimum virtual segment size.





#### USER-MODIFIABLE OPTIONS

- 1) Memory fragmentation - **mmuLib** obtains memory from the system memory manager via **memalign()** to contain the mmu's translation tables. This memory was allocated a page at a time on page boundaries. Unfortunately, in the current memory management scheme, the memory manager is not able to allocate these pages contiguously. Building large translation tables (*i.e.*, when mapping large portions of virtual memory) causes excessive fragmentation of the system memory pool. An attempt to alleviate this has been installed by providing a local buffer of page aligned memory; the user may control the buffer size by manipulating the global variable **mmuNumPagesInFreeList**. By default, **mmuPagesInFreeList** is set to 8.
- 2) Alternate memory source - A customer has special purpose hardware that includes separate static RAM for the mmu's translation tables. Thus, they require the ability to specify an alternate source of memory other than **memalign()**. A global variable has been created that points to the memory partition to be used as the source for translation table memory; by default, it points to the system memory partition. The user may modify this to point to another memory partition before **mmuSh7700LibInit()** is called.

---

## mmuSh7750Lib

NAME	<b>mmuSh7750Lib</b> – Hitachi SH7750 MMU support library
ROUTINES	<b>mmuSh7750LibInit()</b> - initialize module
DESCRIPTION	<b>mmuLib.c</b> provides the architecture dependent routines that directly control the memory management unit. It provides 10 routines that are called by the higher level architecture independent routines in <b>vmLib.c</b> : <ul style="list-style-type: none"><li>– <b>mmuLibInit()</b> - initialize module</li><li>– <b>mmuTransTblCreate()</b> - create a new translation table</li><li>– <b>mmuTransTblDelete()</b> - delete a translation table.</li><li>– <b>mmuEnable()</b> - turn mmu on or off</li><li>– <b>mmuStateSet()</b> - set state of virtual memory page</li><li>– <b>mmuStateGet()</b> - get state of virtual memory page</li><li>– <b>mmuPageMap()</b> - map physical memory page to virtual memory page</li><li>– <b>mmuGlobalPageMap()</b> - map physical memory page to global virtual memory page</li><li>– <b>mmuTranslate()</b> - translate a virtual address to a physical address</li><li>– <b>mmuCurrentSet()</b> - change active translation table</li></ul>

Applications using the mmu will never call these routines directly; the visible interface is supported in **vmLib.c**.

**mmuLib** supports the creation and maintenance of multiple translation tables, one of which is the active translation table when the mmu is enabled. Note that VxWorks does not include a translation table as part of the task context; individual tasks do not reside in private virtual memory. However, we include the facilities to create multiple translation tables so that the user may create “private” virtual memory contexts and switch them in an application specific manner. New translation tables are created with a call to **mmuTransTblCreate()**, and installed as the active translation table with **mmuCurrentSet()**. Translation tables are modified and potentially augmented with calls to **mmuPageMap()** and **mmuStateSet()**. The state of portions of the translation table can be read with calls to **mmuStateGet()** and **mmuTranslate()**.

The traditional VxWorks architecture and design philosophy requires that all objects and operating systems resources be visible and accessible to all agents (tasks, isrs, watchdog timers, *etc.*) in the system. This has traditionally been insured by the fact that all objects and data structures reside in physical memory; thus, a data structure created by one agent may be accessed by any other agent using the same pointer (object identifiers in VxWorks are often pointers to data structures.) This creates a potential problem if you have multiple virtual memory contexts. For example, if a semaphore is created in one virtual memory context, you must guarantee that that semaphore will be visible in all virtual memory contexts if the semaphore is to be accessed at interrupt level, when a virtual memory context other than the one in which it was created may be active. Another example is that code loaded using the incremental loader from the shell must be accessible in all virtual memory contexts, since code is shared by all agents in the system.

This problem is resolved by maintaining a global “transparent” mapping of virtual to physical memory for all the contiguous segments of physical memory (on board memory, i/o space, sections of vme space, *etc.*) that is shared by all translation tables; all available physical memory appears at the same address in virtual memory in all virtual memory contexts. This technique provides an environment that allows resources that rely on a globally accessible physical address to run without modification in a system with multiple virtual memory contexts.

An additional requirement is that modifications made to the state of global virtual memory in one translation table appear in all translation tables. For example, memory containing the text segment is made read only (to avoid accidental corruption) by setting the appropriate writable bits in the translation table entries corresponding to the virtual memory containing the text segment. This state information must be shared by all virtual memory contexts, so that no matter what translation table is active, the text segment is protected from corruption. The mechanism that implements this feature is architecture dependent, but usually entails building a section of a translation table that corresponds to the global memory, that is shared by all other translation tables. Thus, when changes to the state of the global memory are made in one translation table, the changes are reflected in all other translation tables.

**mmuLib** provides a separate call for constructing global virtual memory - **mmuGlobalPageMap()** - which creates translation table entries that are shared by all translation tables. Initialization code in **usrConfig** makes calls to **vmGlobalMap()** (which





The physical memory that holds these data structures is obtained from the system memory manager via **memalign()** to ensure that the memory is page aligned. We want to protect this memory from being corrupted, so we invalidate the descriptors that we set up in the global translation that correspond to the memory containing the translation table data structures. This creates a “chicken and the egg” paradox, in that the only way we can modify these data structures is through virtual memory that is now invalidated, and we can’t validate it because the page descriptors for that memory are in invalidated memory (confused yet?) So, you will notice that anywhere that page table descriptors (ptel’s) are modified, we do so by locking out interrupts, momentarily disabling the mmu, accessing the memory with its physical address, enabling the mmu, and then re-enabling interrupts (see **mmuStateSet()**, for example.)

#### USER MODIFIABLE OPTIONS

- 1) Memory fragmentation - **mmuLib** obtains memory from the system memory manager via **memalign()** to contain the mmu’s translation tables. This memory was allocated a page at a time on page boundaries. Unfortunately, in the current memory management scheme, the memory manager is not able to allocate these pages contiguously. Building large translation tables (*i.e.*, when mapping large portions of virtual memory) causes excessive fragmentation of the system memory pool. An attempt to alleviate this has been installed by providing a local buffer of page aligned memory; the user may control the buffer size by manipulating the global variable **mmuNumPagesInFreeList**. By default, **mmuPagesInFreeList** is set to 8.
- 2) Alternate memory source - A customer has special purpose hardware that includes separate static RAM for the mmu’s translation tables. Thus, they require the ability to specify an alternate source of memory other than **memalign()**. A global variable has been created that points to the memory partition to be used as the source for translation table memory; by default, it points to the system memory partition. The user may modify this to point to another memory partition before **mmuSh7750LibInit()** is called.

---

## moduleLib

**NAME**                    **moduleLib** – object module management library

**ROUTINES**                **moduleCreate()** - create and initialize a module  
**moduleDelete()** - delete module ID information (use **unld()** to reclaim space)  
**moduleShow()** - show the current status for all the loaded modules  
**moduleSegGet()** - get (delete and return) the first segment from a module  
**moduleSegFirst()** - find the first segment in a module  
**moduleSegNext()** - find the next segment in a module  
**moduleCreateHookAdd()** - add a routine to be called when a module is added

**moduleCreateHookDelete()** - delete a previously added module create hook routine  
**moduleFindByName()** - find a module by name  
**moduleFindByNameAndPath()** - find a module by file name and path  
**moduleFindByGroup()** - find a module by group number  
**moduleIdListGet()** - get a list of loaded modules  
**moduleInfoGet()** - get information about an object module  
**moduleCheck()** - verify checksums on all modules  
**moduleNameGet()** - get the name associated with a module ID  
**moduleFlagsGet()** - get the flags associated with a module ID

**DESCRIPTION**

This library is a class manager, using the standard VxWorks class/object facilities. The library is used to keep track of which object modules have been loaded into VxWorks, to maintain information about object module segments associated with each module, and to track which symbols belong to which module. Tracking modules makes it possible to list which modules are currently loaded, and to unload them when they are no longer needed.

The module object contains the following information:

- name
- linked list of segments, including base addresses and sizes
- symbol group number
- format of the object module (a.out, COFF, ECOFF, etc.)
- the *symFlag* passed to **ld()** when the module was loaded. (For more information about *symFlag* and the loader, see the manual entry for **loadLib**.)

Multiple modules with the same name are allowed (the same module may be loaded without first being unloaded) but “find” functions find the most recently created module.

The symbol group number is a unique number for each module, used to identify the module’s symbols in the symbol table. This number is assigned by **moduleLib** when a module is created.

In general, users will not access these routines directly, with the exception of **moduleShow()**, which displays information about currently loaded modules. Most calls to this library will be from routines in **loadLib** and **unldLib**.

**INCLUDE FILES****moduleLib.h****SEE ALSO***loadLib, Tornado User’s Guide: Cross-Development*

---

## mountLib

<b>NAME</b>	<b>mountLib</b> – mount protocol library
<b>ROUTINES</b>	<b>mountdInit()</b> - initialize the mount daemon <b>nfsExport()</b> - specify a file system to be NFS exported <b>nfsUnexport()</b> - remove a file system from the list of exported file systems
<b>DESCRIPTION</b>	This library implements a mount server to support mounting VxWorks file systems remotely. The mount server is an implementation of version 1 of the mount protocol as defined in RFC 1094. It is closely connected with version 2 of the Network File System Protocol Specification, which in turn is implemented by the library <b>nfsdLib</b> .

---

**NOTE:** The only routines in this library that are normally called by applications are **nfsExport()** and **nfsUnexport()**. The mount daemon is normally initialized indirectly by **nfsdInit()**.

---

The mount server is initialized by calling **mountdInit()**. Normally, this is done by **nfsdInit()**, although it is possible to call **mountdInit()** directly if the NFS server is not being initialized. Defining **INCLUDE\_NFS\_SERVER** enables the call to **nfsdInit()** during the boot process, which in turn calls **mountdInit()**, so there is normally no need to call either routine manually. **mountdInit()** spawns one task, **tMountd**, which registers as an RPC service with the portmapper.

Currently, only the **dosFsLib** file system is supported. File systems are exported with the **nfsExport()** call.

To export VxWorks file systems via NFS, you need facilities from both this library and from **nfsdLib**. To include both, add **INCLUDE\_NFS\_SERVER** and rebuild VxWorks.

### Example

The following example illustrates how to export an existing dosFs file system.

First, initialize the block device containing your file system.

Then assuming the dosFs system is called **/export** execute the following code on the target:

```
nfsExport ("/export", 0, FALSE, 0);          /* make available remotely */
```

This makes it available to all clients to be mounted using the client's NFS mounting command. (On UNIX systems, mounting file systems normally requires root privileges.)

VxWorks does not normally provide authentication services for NFS requests, and the DOS file system does not provide file permissions. If you need to authenticate incoming requests, see the documentation for **nfsdInit()** and **mountdInit()** for information about authorization hooks.

The following requests are accepted from clients. For details of their use, see Appendix A of RFC 1094, “NFS: Network File System Protocol Specification.”

<b>Procedure Name</b>	<b>Procedure Number</b>
MOUNTPROC_NULL	0
MOUNTPROC_MNT	1
MOUNTPROC_DUMP	2
MOUNTPROC_UMNT	3
MOUNTPROC_UMNTALL	4
MOUNTPROC_EXPORT	5

**SEE ALSO** `dosFsLib`, `nfsdLib`, RFC 1094

---

## mqPxLib

**NAME** `mqPxLib` – message queue library (POSIX)

**ROUTINES**

- `mqPxLibInit()` - initialize the POSIX message queue library
- `mq_open()` - open a message queue (POSIX)
- `mq_receive()` - receive a message from a message queue (POSIX)
- `mq_send()` - send a message to a message queue (POSIX)
- `mq_close()` - close a message queue (POSIX)
- `mq_unlink()` - remove a message queue (POSIX)
- `mq_notify()` - notify a task that a message is available on a queue (POSIX)
- `mq_setattr()` - set message queue attributes (POSIX)
- `mq_getattr()` - get message queue attributes (POSIX)

**DESCRIPTION** This library implements the message-queue interface defined in the POSIX 1003.1b standard, as an alternative to the VxWorks-specific message queue design in `msgQLib`. These message queues are accessed through names; each message queue supports multiple sending and receiving tasks.

The message queue interface imposes a fixed upper bound on the size of messages that can be sent to a specific message queue. The size is set on an individual queue basis. The value may not be changed dynamically.

This interface allows a task be notified asynchronously of the availability of a message on the queue. The purpose of this feature is to let the task to perform other functions and yet still be notified that a message has become available on the queue.

#### MESSAGE QUEUE DESCRIPTOR DELETION

The `mq_close()` call terminates a message queue descriptor and deallocates any associated memory. When deleting message queue descriptors, take care to avoid interfering with other tasks that are using the same descriptor. Tasks should only close message queue descriptors that the same task has opened successfully.

The routines in this library conform to POSIX 1003.1b.

**INCLUDE FILES**     `mqueue.h`

**SEE ALSO**         POSIX 1003.1b document, `msgQLib`, *VxWorks Programmer's Guide: Basic OS*

---

## mqPxShow

**NAME**             `mqPxShow` – POSIX message queue show

**ROUTINES**        `mqPxShowInit()` - initialize the POSIX message queue show facility

**DESCRIPTION**     This library provides a show routine for POSIX objects.

---

## msgQDistGrpLib

**NAME**             `msgQDistGrpLib` – distributed message queue group library (VxFusion Opt.)

**ROUTINES**        `msgQDistGrpAdd()` - add a distributed message queue to a group (VxFusion Opt.)  
`msgQDistGrpDelete()` - delete a distributed message queue from a group (VxFusion Opt.)

**DESCRIPTION**     This library provides the grouping facility for distributed message queues. Single distributed message queues can join one or more groups. A message sent to a group is sent to all message queues that are members of that group. A group, however, is prohibited from sending messages. Also, it is an error to call `msgQDistNumMsgs()` with a distributed message queue group ID.

Groups are created with symbolic names and identified by a unique ID, `MSG_Q_ID`, as with normal message queues.

If the group is new to the distributed system, the group agreement protocol (GAP) is employed to determine a globally unique identifier. As part of the protocol's negotiation, all group databases throughout the system are updated.

The distributed message queue group library is initialized by calling **distInit()**.

<b>AVAILABILITY</b>	This module is distributed as a component of the unbundled distributed message queues option, VxFusion.
<b>INCLUDE FILES</b>	<b>msgQDistGrpLib.h</b>
<b>SEE ALSO</b>	<b>distLib, msgQDistGrpShow</b>

---

## msgQDistGrpShow

<b>NAME</b>	<b>msgQDistGrpShow</b> – distributed message queue group show routines (VxFusion Opt.)
<b>ROUTINES</b>	<b>msgQDistGrpShow()</b> - display all or one group with its members (VxFusion Opt.)
<b>DESCRIPTION</b>	This library provides a routine to show either the contents of the entire message queue group database or the contents of single message queue group.
<b>AVAILABILITY</b>	This module is distributed as a component of the unbundled distributed message queues option, VxFusion.
<b>INCLUDE FILES</b>	<b>msgQDistGrpShow.h</b>
<b>SEE ALSO</b>	<b>msgQDistGrpLib</b>

---

## msgQDistLib

<b>NAME</b>	<b>msgQDistLib</b> – distributed objects message queue library (VxFusion Opt.)
<b>ROUTINES</b>	<b>msgQDistCreate()</b> - create a distributed message queue (VxFusion Opt.) <b>msgQDistSend()</b> - send a message to a distributed message queue (VxFusion Opt.) <b>msgQDistReceive()</b> - receive a message from a distributed message queue (VxFusion Opt.) <b>msgQDistNumMsgs()</b> - get the number of messages in a distributed message queue (VxFusion Opt.)
<b>DESCRIPTION</b>	This library provides the interface to distributed message queues. Any task on any node in the system can send messages to or receive from a distributed message queue. Full



duplex communication between two tasks generally requires two distributed message queues, one for each direction.

Distributed message queues are created with **msgQDistCreate()**. After creation, they can be manipulated using the generic routines for local message queues; for more information on the use of these routines, see the manual entry for **msgQLib**. The **msgQDistLib** library also provides the **msgQDistSend()**, **msgQDistReceive()**, and **msgQDistNumMsgs()** routines which support additional parameters that are useful for working with distributed message queues.

The distributed objects message queue library is initialized by calling **distInit()**.

<b>AVAILABILITY</b>	This module is distributed as a component of the unbundled distributed message queues option, VxFusion.
<b>INCLUDE FILES</b>	<b>msgQDistLib.h</b>
<b>SEE ALSO</b>	<b>msgQLib, msgQDistShow, distLib</b>

---

## msgQDistShow

<b>NAME</b>	<b>msgQDistShow</b> – distributed message queue show routines (VxFusion Opt.)
<b>ROUTINES</b>	<b>msgQDistShowInit()</b> - initialize the distributed message queue show package (VxFusion Opt.)
<b>DESCRIPTION</b>	This library provides show routines for distributed message queues. The user does not call these show routines directly. Instead, he uses the <b>msgQShow</b> library routine <b>msgQShow()</b> to display the contents of a message queue, regardless of its type. The <b>msgQShow()</b> routine calls the distributed show routines, as necessary.
<b>AVAILABILITY</b>	This module is distributed as a component of the unbundled distributed message queues option, VxFusion.
<b>INCLUDE FILES</b>	<b>msgQDistShow.h</b>
<b>SEE ALSO</b>	<b>msgQDistLib, msgQShow</b>

---

## msgQEvLib

<b>NAME</b>	<b>msgQEvLib</b> – VxWorks events support for message queues
<b>ROUTINES</b>	<b>msgQEvStart()</b> - start event notification process for a message queue <b>msgQEvStop()</b> - stop event notification process for a message queue
<b>DESCRIPTION</b>	<p>This library is an extension to <b>eventLib</b>, the events library. Its purpose is to support events for message queues.</p> <p>The functions in this library are used to control registration of tasks on a message queue. The routine <b>msgQEvStart()</b> registers a task and starts the notification process. The function <b>msgQEvStop()</b> un-registers the task, which stops the notification mechanism.</p> <p>When a task is registered and a message arrives on the queue, the events specified are sent to that task, on the condition that no other task is pending on that message queue. However, if a <b>msgQReceive()</b> is to be done afterwards to get the message, there is no guarantee that it will still be available.</p>
<b>INCLUDE FILES</b>	<b>msgQEvLib.h</b>
<b>SEE ALSO</b>	<b>eventLib</b> , <i>VxWorks Programmer's Guide: Basic OS</i>

---

## msgQLib

<b>NAME</b>	<b>msgQLib</b> – message queue library
<b>ROUTINES</b>	<b>msgQCreate()</b> - create and initialize a message queue <b>msgQDelete()</b> - delete a message queue <b>msgQSend()</b> - send a message to a message queue <b>msgQReceive()</b> - receive a message from a message queue <b>msgQNumMsgs()</b> - get the number of messages queued to a message queue
<b>DESCRIPTION</b>	<p>This library contains routines for creating and using message queues, the primary intertask communication mechanism within a single CPU. Message queues allow a variable number of messages (varying in length) to be queued in first-in-first-out (FIFO) order. Any task or interrupt service routine can send messages to a message queue. Any task can receive messages from a message queue. Multiple tasks can send to and receive from the same message queue. Full-duplex communication between two tasks generally requires two message queues, one for each direction.</p>

To provide message queue support for a system, VxWorks must be configured with the `INCLUDE_MSG_Q` component.

#### CREATING AND USING MESSAGE QUEUES

A message queue is created with `msgQCreate()`. Its parameters specify the maximum number of messages that can be queued to that message queue and the maximum length in bytes of each message. Enough buffer space will be pre-allocated to accommodate the specified number of messages of specified length.

A task or interrupt service routine sends a message to a message queue with `msgQSend()`. If no tasks are waiting for messages on the message queue, the message is simply added to the buffer of messages for that queue. If any tasks are already waiting to receive a message from the message queue, the message is immediately delivered to the first waiting task.

A task receives a message from a message queue with `msgQReceive()`. If any messages are already available in the message queue's buffer, the first message is immediately dequeued and returned to the caller. If no messages are available, the calling task will block and be added to a queue of tasks waiting for messages. This queue of waiting tasks can be ordered either by task priority or FIFO, as specified in an option parameter when the queue is created.

#### TIMEOUTS

Both `msgQSend()` and `msgQReceive()` take timeout parameters. When sending a message, if no buffer space is available to queue the message, the timeout specifies how many ticks to wait for space to become available. When receiving a message, the timeout specifies how many ticks to wait if no message is immediately available. The *timeout* parameter can have the special values `NO_WAIT` (0) or `WAIT_FOREVER` (-1). `NO_WAIT` means the routine should return immediately; `WAIT_FOREVER` means the routine should never time out.

#### URGENT MESSAGES

The `msgQSend()` routine allows the priority of a message to be specified as either normal or urgent, `MSG_PRI_NORMAL` (0) and `MSG_PRI_URGENT` (1), respectively. Normal priority messages are added to the tail of the list of queued messages, while urgent priority messages are added to the head of the list.

**VXWORKS EVENTS** If a task has registered with a message queue via `msgQEvStart()`, events will be sent to that task when a message arrives on that message queue, on the condition that no other task is pending on the queue.

**INCLUDE FILES** `msgQLib.h`

**SEE ALSO** `pipeDrv`, `msgQSmLib`, `msgQEvLib`, `eventLib`, *VxWorks Programmer's Guide: Basic OS*

---

## msgQShow

<b>NAME</b>	<b>msgQShow</b> – message queue show routines
<b>ROUTINES</b>	<b>msgQShowInit()</b> - initialize the message queue show facility <b>msgQInfoGet()</b> - get information about a message queue <b>msgQShow()</b> - show information about a message queue
<b>DESCRIPTION</b>	<p>This library provides routines to show message queue statistics, such as the task queuing method, messages queued, receivers blocked, <i>etc.</i></p> <p>The routine <b>msgQshowInit()</b> links the message queue show facility into the VxWorks system. It is called automatically when the message queue show facility is configured into VxWorks using either of the following methods:</p> <p>If you use the configuration header files, define <b>INCLUDE_SHOW_ROUTINES</b> in <b>config.h</b>.</p> <p>If you use the Tornado project facility, select <b>INCLUDE_MSG_Q_SHOW</b>.</p>
<b>INCLUDE FILES</b>	<b>msgQLib.h</b>
<b>SEE ALSO</b>	<b>pipeDrv</b> , <i>VxWorks Programmer's Guide: Basic OS</i>

---

## msgQSmLib

<b>NAME</b>	<b>msgQSmLib</b> – shared memory message queue library (VxMP Opt.)
<b>ROUTINES</b>	<b>msgQSmCreate()</b> - create and initialize a shared memory message queue (VxMP Opt.)
<b>DESCRIPTION</b>	<p>This library provides the interface to shared memory message queues. Shared memory message queues allow a variable number of messages (varying in length) to be queued in first-in-first-out order. Any task running on any CPU in the system can send messages to or receive messages from a shared message queue. Tasks can also send to and receive from the same shared message queue. Full-duplex communication between two tasks generally requires two shared message queues, one for each direction.</p> <p>Shared memory message queues are created with <b>msgQSmCreate()</b>. Once created, they can be manipulated using the generic routines for local message queues; for more information on the use of these routines, see the manual entry for <b>msgQLib</b>.</p>
<b>MEMORY REQUIREMENTS</b>	The shared memory message queue structure is allocated from a dedicated shared

**muxLib**

memory partition. This shared memory partition is initialized by the shared memory objects master CPU. The size of this partition is defined by the maximum number of shared message queues, **SM\_OBJ\_MAX\_MSG\_Q**.

The message queue buffers are allocated from the shared memory system partition.

**RESTRICTIONS** Shared memory message queues differ from local message queues in the following ways:

**Interrupt Use:**

Shared memory message queues may not be used (sent to or received from) at interrupt level.

**Deletion:**

There is no way to delete a shared memory message queue and free its associated shared memory. Attempts to delete a shared message queue return **ERROR** and set **errno** to **S\_smObjLib\_NO\_OBJECT\_DESTROY**.

**Queuing Style:**

The shared message queue task queuing order specified when a message queue is created must be FIFO.

**CONFIGURATION** Before routines in this library can be called, the shared memory objects facility must be initialized by calling **usrSmObjInit()**. This is done automatically during VxWorks initialization if the component **INCLUDE\_SM\_OBJ** is included.

**AVAILABILITY** This module is distributed as a component of the unbundled shared objects memory support option, VxMP.

**INCLUDE FILES** **msgQSmLib.h, msgQLib.h, smMemLib.h, smObjLib.h**

**SEE ALSO** **msgQLib, smObjLib, msgQShow, usrSmObjInit()**, *VxWorks Programmer's Guide: Shared Memory Objects*

---

## muxLib

**NAME** **muxLib** – MUX network interface library

**ROUTINES**

- muxLibInit()** - initialize global state for the MUX
- muxDevLoad()** - load a driver into the MUX
- muxDevStart()** - start a device by calling its start routine
- muxDevStop()** - stop a device by calling its stop routine
- muxShow()** - display configuration of devices registered with the MUX
- muxBind()** - create a binding between a network service and an END
- muxSend()** - send a packet out on a network interface

**muxPollSend()** - now **deprecated**, see **muxTkPollSend()**  
**muxPollReceive()** - now **deprecated**, see **muxTkPollReceive()**  
**muxIoctl()** - send control information to the MUX or to a device  
**muxMCastAddrAdd()** - add a multicast address to a device's multicast table  
**muxMCastAddrDel()** - delete a multicast address from a device's multicast table  
**muxMCastAddrGet()** - get the multicast address table from the MUX/Driver  
**muxUnbind()** - detach a network service from the specified device  
**muxDevUnload()** - unloads a device from the MUX  
**muxLinkHeaderCreate()** - attach a link-level header to a packet  
**muxAddressForm()** - form a frame with a link-layer address  
**muxPacketDataGet()** - return the data from a packet  
**muxPacketAddrGet()** - get addressing information from a packet  
**endFindByName()** - find a device using its string name  
**muxDevExists()** - tests whether a device is already loaded into the MUX  
**muxAddrResFuncAdd()** - replace the default address resolution function  
**muxAddrResFuncGet()** - get the address resolution function for ifType/protocol  
**muxAddrResFuncDel()** - delete an address resolution function  
**muxTaskDelaySet()** - set the inter-cycle delay on the polling task  
**muxTaskDelayGet()** - get the delay on the polling task  
**muxTaskPrioritySet()** - reset the priority of **tMuxPollTask**  
**muxTaskPriorityGet()** - get the priority of **tMuxPollTask**  
**muxPollStart()** - initialize and start the MUX poll task  
**muxPollEnd()** - shuts down **tMuxPollTask** and returns devices to interrupt mode  
**muxPollDevAdd()** - adds a device to list polled by **tMuxPollTask**  
**muxPollDevDel()** - removes a device from the list polled by **tMuxPollTask**  
**muxPollDevStat()** - reports whether device is on list polled by **tMuxPollTask**

**DESCRIPTION**

This library provides the routines that define the MUX interface, a facility that handles communication between the data link layer and the network protocol layer. Using the MUX, the VxWorks network stack has decoupled the data link and network layers. Drivers and services no longer need knowledge of each other's internals. This independence makes it much easier to add new drivers or services. For example, if you add a new MUX-based "END" driver, all existing MUX-based services can use the new driver. Likewise, if you add a new MUX-based service, any existing END can use the MUX to access the new service.

**INCLUDE FILES**

**errno.h**, **lstLib.h**, **logLib.h**, **string.h**, **m2Lib.h**, **bufLib.h**, **if.h**, **end.h**, **muxLib.h**, **vxWorks.h**, **taskLib.h**, **stdio.h**, **errnoLib.h**, **if\_ether.h**, **netLib.h**, **semLib.h**, **rebootLib.h**

To use this feature, include the following component: **INCLUDE\_MUX**

**SEE ALSO**

*VxWorks AE Network Programmer's Guide*

**M**

## muxTkLib

<b>NAME</b>	<b>muxTkLib</b> – MUX toolkit Network Interface Library
<b>ROUTINES</b>	<b>muxTkDrvCheck()</b> - checks if the device is an NPT or an END interface <b>muxTkCookieGet()</b> - returns the cookie for a device <b>muxTkBind()</b> - bind an NPT protocol to a driver <b>muxTkReceive()</b> - receive a packet from a NPT driver <b>muxTkSend()</b> - send a packet out on a Toolkit or END network interface <b>muxTkPollSend()</b> - send a packet out in polled mode to an END or NPT interface <b>muxTkPollReceive()</b> - poll for a packet from a NPT or END driver
<b>DESCRIPTION</b>	<p>This library provides additional APIs offered by the Network Protocol Toolkit (NPT) architecture. These APIs extend the original release of the MUX interface.</p> <p>A NPT driver is an enhanced END but retains all of the END's functionality. NPT also introduces the term "network service sublayer" or simply "service sublayer" which is the component that interfaces between the network service (or network protocol) and the MUX. This service sublayer may be built in to the network service or protocol rather than being a separate component.</p>
<b>INCLUDE FILES</b>	<b>vxWorks.h, taskLib.h, stdio.h, errno.h, herrnoLib.h, lstlib.h, logLib.h, string.h, m2Lib.h, net/if.h, bufLib.h, semlib.h, end.h, muxLib.h, muxTkLib.h, netinet/if_ether.h, net/mbuf.h</b>

---

## netBufLib

<b>NAME</b>	<b>netBufLib</b> – network buffer library
<b>ROUTINES</b>	<p><b>netBufLibInit()</b> - initialize <b>netBufLib</b></p> <p><b>netPoolInit()</b> - initialize a <b>netBufLib</b>-managed memory pool</p> <p><b>netPoolKheapInit()</b> - kernel heap version of <b>netPoolInit()</b></p> <p><b>netPoolDelete()</b> - delete a memory pool</p> <p><b>netMblkFree()</b> - free an <b>mBlk</b> back to its memory pool</p> <p><b>netCIBlkFree()</b> - free a <b>cBlk</b>-cluster construct back to the memory pool</p> <p><b>netCIFree()</b> - free a cluster back to the memory pool</p> <p><b>netMblkCIFree()</b> - free an <b>mBlk-cBlk</b>-cluster construct</p> <p><b>netMblkCIChainFree()</b> - free a chain of <b>mBlk-cBlk</b>-cluster constructs</p> <p><b>netMblkGet()</b> - get an <b>mBlk</b> from a memory pool</p> <p><b>netCIBlkGet()</b> - get a <b>cBlk</b></p> <p><b>netClusterGet()</b> - get a cluster from the specified cluster pool</p> <p><b>netMblkCIGet()</b> - get a <b>cBlk</b>-cluster and join it to the specified <b>mBlk</b></p> <p><b>netTupleGet()</b> - get an <b>mBlk-cBlk</b>-cluster</p> <p><b>netCIBlkJoin()</b> - join a cluster to a <b>cBlk</b> structure</p> <p><b>netMblkCIJoin()</b> - join an <b>mBlk</b> to a <b>cBlk</b>-cluster construct</p> <p><b>netCIPoolIdGet()</b> - return a <b>CL_POOL_ID</b> for a specified buffer size</p> <p><b>netMblkToBufCopy()</b> - copy data from an <b>mBlk</b> to a buffer</p> <p><b>netMblkDup()</b> - duplicate an <b>mBlk</b></p> <p><b>netMblkChainDup()</b> - duplicate an <b>mBlk</b> chain</p>
<b>DESCRIPTION</b>	<p>This library contains routines that you can use to organize and maintain a memory pool that consists of pools of <b>mBlk</b> structures, pools of <b>cBlk</b> structures, and pools of clusters. The <b>mBlk</b> and <b>cBlk</b> structures are used to manage the clusters. The clusters are containers for the data described by the <b>mBlk</b> and <b>cBlk</b> structures.</p> <p>These structures and the various routines of this library constitute a buffering API that has been designed to meet the needs both of network protocols and network device drivers.</p> <p>The <b>mBlk</b> structure is the primary vehicle for passing data between a network driver and a protocol. However, the <b>mBlk</b> structure must first be properly joined with a <b>cBlk</b> structure that was previously joined with a cluster. Thus, the actual vehicle for passing data is not merely an <b>mBlk</b> structure but an <b>mBlk-cBlk</b>-cluster construct.</p> <p>To use this feature, include the following component: <b>INCLUDE_NETWRS_NETBUFLIB</b></p>
<b>INCLUDE FILES</b>	<b>netBufLib.h</b>

## netDrv

- NAME** **netDrv** – network remote file I/O driver
- ROUTINES** **netDrv()** - install the network remote file driver  
**netDevCreate()** - create a remote file device  
**netDevCreate2()** - create a remote file device with fixed buffer size  
**netDrvDebugLevelSet()** - set the debug level of the **netDrv** library routines  
**netDrvFileDoesNotExistInstall()** - install an applet to test if a file exists
- DESCRIPTION** This driver provides facilities for accessing files transparently over the network via FTP or RSH. By creating a network device with **netDevCreate()**, files on a remote UNIX machine may be accessed as if they were local.
- When a remote file is opened, the entire file is copied over the network to a local buffer. When a remote file is created, an empty local buffer is opened. Any reads, writes, or **ioctl()** calls are performed on the local copy of the file. If the file was opened with the flags **O\_WRONLY** or **O\_RDWR** and modified, the local copy is sent back over the network to the UNIX machine when the file is closed.
- Note that this copying of the entire file back and forth can make **netDrv** devices awkward to use. A preferable mechanism is NFS as provided by **nfsDrv**.
- USER-CALLABLE ROUTINES**
- Most of the routines in this driver are accessible only through the I/O system. However, two routines must be called directly: **netDrv()** to initialize the driver and **netDevCreate()** to create devices.
- FILE OPERATIONS** This driver supports the creation, deletion, opening, reading, writing, and appending of files. The renaming of files is not supported.
- INITIALIZATION** Before using the driver, it must be initialized by calling the routine **netDrv()**. This routine should be called only once, before any reads, writes, **netDevCreate()**, or **netDevCreate2()** calls. Initialization is performed automatically when **INCLUDE\_NET\_DRV** is defined.

### CREATING NETWORK DEVICES

To access files on a remote host, a network device must be created by calling **netDevCreate()** or **netDevCreate2()**. The arguments to **netDevCreate()** are the name of the device, the name of the host the device will access, and the remote file access protocol to be used -- RSH or FTP. The arguments to **netDevCreate2()** are ones described above and a size of buffer used in the network device as a fourth argument. By convention, a network device name is the remote machine name followed by a colon ":". For example, for a UNIX host on the network "wrs", files can be accessed by creating a device called "wrs:". For more information, see the manual entry for **netDevCreate()** and **netDevCreate2()**.

**IOCTL FUNCTIONS** The network driver responds to the following `ioctl()` functions:

**FIOGETNAME**

Gets the file name of the file descriptor `fd` and copies it to the buffer specified by `nameBuf`:

```
status = ioctl (fd, FIOGETNAME, &nameBuf);
```

**FIONREAD**

Copies to `nBytesUnread` the number of bytes remaining in the file specified by `fd`:

```
status = ioctl (fd, FIONREAD, &nBytesUnread);
```

**FIOSEEK**

Sets the current byte offset in the file to the position specified by `newOffset`. If the seek goes beyond the end-of-file, the file grows. The end-of-file pointer changes to the new position, and the new space is filled with zeroes:

```
status = ioctl (fd, FIOSEEK, newOffset);
```

**FIOWHERE**

Returns the current byte position in the file. This is the byte offset of the next byte to be read or written. It takes no additional argument:

```
position = ioctl (fd, FIOWHERE, 0);
```

**FIOFSTATGET**

Gets file status information. The argument `statStruct` is a pointer to a `stat` structure that is filled with data describing the specified file. Normally, the `stat()` or `fstat()` routine is used to obtain file information, rather than using the `FIOFSTATGET` function directly. `netDrv` only fills in three fields of the `stat` structure: `st_dev`, `st_mode`, and `st_size`. `st_mode` is always filled with `S_IFREG`.

```
struct stat statStruct;
fd = open ("file", O_RDONLY);
status = ioctl (fd, FIOFSTATGET, &statStruct);
```

**LIMITATIONS**

The `netDrv` implementation strategy implies that directories cannot always be distinguished from plain files. Thus, `opendir()` does not work for directories mounted on `netDrv` devices, and `lstat()` does not flag subdirectories with the label "DIR" in listings from `netDrv` devices.

When the access method is FTP, operations can only be done on files that the FTP server allows to download. In particular it is not possible to `stat` a directory, doing so will result in "dirname: not a plain file" error.

**INCLUDE FILES**

`netDrv.h`

**SEE ALSO**

`remLib`, `netLib`, `sockLib`, `hostAdd()`

## netLib

<b>NAME</b>	<b>netLib</b> – network interface library
<b>ROUTINES</b>	<b>netLibInit()</b> - initialize the network package <b>netTask()</b> - network task entry point
<b>DESCRIPTION</b>	<p>This library contains the network task that runs low-level network interface routines in a task context. The network task executes and removes routines that were added to the job queue. This facility is used by network interfaces in order to have interrupt-level processing at task level.</p> <p>The routine <b>netLibInit()</b> initializes the network and spawns the network task <b>netTask()</b>. This is done automatically when <b>INCLUDE_NET_LIB</b> is defined.</p> <p>The routine <b>netHelp()</b> in <b>usrLib</b> displays a summary of the network facilities available from the VxWorks shell.</p>
<b>INCLUDE FILES</b>	<b>netLib.h</b>
<b>SEE ALSO</b>	<b>routeLib</b> , <b>hostLib</b> , <b>netDrv</b> , <b>netHelp()</b> ,

---

## netShow

<b>NAME</b>	<b>netShow</b> – network information display routines
<b>ROUTINES</b>	<b>ifShow()</b> - display the attached network interfaces <b>inetstatShow()</b> - display all active connections for Internet protocol sockets <b>ipstatShow()</b> - display IP statistics <b>netPoolShow()</b> - show pool statistics <b>netStackDataPoolShow()</b> - show network stack data pool statistics <b>netStackSysPoolShow()</b> - show network stack system pool statistics <b>mbufShow()</b> - report <b>mbuf</b> statistics <b>netShowInit()</b> - initialize network show routines <b>arpShow()</b> - display entries in the system ARP table <b>arptabShow()</b> - display the known ARP entries <b>routestatShow()</b> - display routing statistics <b>routeShow()</b> - display all IP routes (summary information) <b>hostShow()</b> - display the host table <b>mRouteShow()</b> - display all IP routes (verbose information)

<b>DESCRIPTION</b>	<p>This library provides routines to show various network-related statistics, such as configuration parameters for network interfaces, protocol statistics, socket statistics, and so on.</p> <p>Interpreting these statistics requires detailed knowledge of Internet network protocols. Information on these protocols can be found in the following books:</p> <p><i>Internetworking with TCP/IP Volume III</i>, by Douglas Comer and David Stevens</p> <p><i>UNIX Network Programming</i>, by Richard Stevens</p> <p><i>The Design and Implementation of the 4.3 BSD UNIX Operating System</i>, by Leffler, McKusick, Karels and Quarterman</p> <p>The <b>netShowInit()</b> routine links the network show facility into the VxWorks system. This is performed automatically if <b>INCLUDE_NET_SHOW</b> is defined. If you want <b>inetstatShow()</b> to display TCP socket status, then <b>INCLUDE_TCP_SHOW</b> needs to be included.</p>
<b>SEE ALSO</b>	<b>ifLib, icmpShow, igmpShow, tcpShow, udpShow</b>

---

## nfsdLib

<b>NAME</b>	<b>nfsdLib</b> – Network File System (NFS) server library
<b>ROUTINES</b>	<b>nfsdInit()</b> - initialize the NFS server <b>nfsdStatusGet()</b> - get the status of the NFS server <b>nfsdStatusShow()</b> - show the status of the NFS server
<b>DESCRIPTION</b>	<p>This library is an implementation of version 2 of the Network File System Protocol Specification as defined in RFC 1094. It is closely connected with version 1 of the mount protocol, also defined in RFC 1094 and implemented in turn by <b>mountLib</b>.</p> <p>The NFS server is initialized by calling <b>nfsdInit()</b>. This is done automatically at boot time if <b>INCLUDE_NFS_SERVER</b> is defined.</p> <p>Currently, only the <b>dosFsLib</b> file system is supported. File systems are exported with the <b>nfsExport()</b> call.</p> <p>To create and export a file system, define <b>INCLUDE_NFS_SERVER</b> and rebuild VxWorks.</p> <p>To export VxWorks file systems via NFS, you need facilities from both this library and from <b>mountLib</b>. To include both, define <b>INCLUDE_NFS_SERVER</b> and rebuild VxWorks.</p> <p>Use the <b>mountLib</b> routine <b>nfsExport()</b> to export file systems. For an example, see the manual page for <b>mountLib</b>.</p>

VxWorks does not normally provide authentication services for NFS requests, and the DOS file system does not provide file permissions. If you need to authenticate incoming requests, see the documentation for **nfsdInit()** and **mountdInit()** for information about authorization hooks.

The following requests are accepted from clients. For details of their use, see RFC 1094, "NFS: Network File System Protocol Specification."

<b>Procedure Name</b>	<b>Procedure Number</b>
NFSPROC_NULL	0
NFSPROC_GETATTR	1
NFSPROC_SETATTR	2
NFSPROC_ROOT	3
NFSPROC_LOOKUP	4
NFSPROC_READLINK	5
NFSPROC_READ	6
NFSPROC_WRITE	8
NFSPROC_CREATE	9
NFSPROC_REMOVE	10
NFSPROC_RENAME	11
NFSPROC_LINK	12
NFSPROC_SYMLINK	13
NFSPROC_MKDIR	14
NFSPROC_RMDIR	15
NFSPROC_READDIR	16
NFSPROC_STATFS	17

#### **AUTHENTICATION AND PERMISSIONS**

Currently, no authentication is done on NFS requests. **nfsdInit()** describes the authentication hooks that can be added should authentication be necessary.

Note that the DOS file system does not provide information about ownership or permissions on individual files. Before initializing a dosFs file system, three global variables--**dosFsUserId**, **dosFsGroupId**, and **dosFsFileMode**--can be set to define the user ID, group ID, and permissions byte for all files in all dosFs volumes initialized after setting these variables. To arrange for different dosFs volumes to use different user and group ID numbers, reset these variables before each volume is initialized. See the manual entry for **dosFsLib** for more information.

#### **TASKS**

Several NFS tasks are created by **nfsdInit()**. They are:

##### **tMountd**

The mount daemon, which handles all incoming mount requests. This daemon is created by **mountdInit()**, which is automatically called from **nfsdInit()**.

**tNfsd**

The NFS daemon, which queues all incoming NFS requests.

**tNfsdX**

The NFS request handlers, which dequeues and processes all incoming NFS requests.

Performance of the NFS file system can be improved by increasing the number of servers specified in the **nfsdInit()** call, if there are several different dosFs volumes exported from the same target system. The **spy()** utility can be called to determine whether this is useful for a particular configuration.

---

## nfsDrv

<b>NAME</b>	<b>nfsDrv</b> – Network File System (NFS) I/O driver
<b>ROUTINES</b>	<b>nfsDrv()</b> - install the NFS driver <b>nfsDrvNumGet()</b> - return the IO system driver number for the NFS driver <b>nfsMount()</b> - mount an NFS file system <b>nfsMountAll()</b> - mount all file systems exported by a specified host <b>nfsDevShow()</b> - display the mounted NFS devices <b>nfsUnmount()</b> - unmount an NFS device <b>nfsDevListGet()</b> - create list of all the NFS devices in the system <b>nfsDevInfoGet()</b> - read configuration information from the requested NFS device
<b>DESCRIPTION</b>	This driver provides facilities for accessing files transparently over the network via NFS (Network File System). By creating a network device with <b>nfsMount()</b> , files on a remote NFS system (such as a UNIX system) can be handled as if they were local.
<b>USER-CALLABLE ROUTINES</b>	The <b>nfsDrv()</b> routine initializes the driver. The <b>nfsMount()</b> and <b>nfsUnmount()</b> routines mount and unmount file systems. The <b>nfsMountAll()</b> routine mounts all file systems exported by a specified host.
<b>INITIALIZATION</b>	Before using the network driver, it must be initialized by calling <b>nfsDrv()</b> . This routine must be called before any reads, writes, or other NFS calls. This is done automatically when <b>INCLUDE_NFS</b> is defined.
<b>CREATING NFS DEVICES</b>	In order to access a remote file system, an NFS device must be created by calling <b>nfsMount()</b> . For example, to create the device <b>/myd0/</b> for the file system <b>/d0/</b> on the host <b>wrs</b> , call:

```
nfsMount ("wrs", "/d0/", "/myd0/");
```

The file `/d0/dog` on the host `wrs` can now be accessed as `/myd0/dog`.

If the third parameter to `nfsMount()` is `NULL`, VxWorks creates a device with the same name as the file system. For example, the call:

```
nfsMount ("wrs", "/d0/", NULL);
```

or from the shell:

```
nfsMount "wrs", "/d0/"
```

creates the device `/d0/`. The file `/d0/dog` is accessed by the same name, `/d0/dog`.

Before mounting a file system, the host must already have been created with `hostAdd()`. The routine `nfsDevShow()` displays the mounted NFS devices.

**IOCTL FUNCTIONS** The NFS driver responds to the following `ioctl()` functions:

#### FIOGETNAME

Gets the file name of `fd` and copies it to the buffer referenced by `nameBuf`:

```
status = ioctl (fd, FIOGETNAME, &nameBuf);
```

#### FIONREAD

Copies to `nBytesUnread` the number of bytes remaining in the file specified by `fd`:

```
status = ioctl (fd, FIONREAD, &nBytesUnread);
```

#### FIOSEEK

Sets the current byte offset in the file to the position specified by `newOffset`. If the seek goes beyond the end-of-file, the file grows. The end-of-file pointer gets moved to the new position, and the new space is filled with zeros:

```
status = ioctl (fd, FIOSEEK, newOffset);
```

#### FIOSYNC

Flush data to the remote NFS file. It takes no additional argument:

```
status = ioctl (fd, FIOSYNC, 0);
```

#### FIOWHERE

Returns the current byte position in the file. This is the byte offset of the next byte to be read or written. It takes no additional argument:

```
position = ioctl (fd, FIOWHERE, 0);
```

#### FIOREADDIR

Reads the next directory entry. The argument `dirStruct` is a pointer to a directory descriptor of type `DIR`. Normally, the `readdir()` routine is used to read a directory, rather than using the `FIOREADDIR` function directly. See the manual entry for `dirLib`:

```
DIR dirStruct;  
fd = open ("directory", O_RDONLY);  
status = ioctl (fd, FIOREADDIR, &dirStruct);
```

**FIOFSTATGET**

Gets file status information (directory entry data). The argument *statStruct* is a pointer to a *stat* structure that is filled with data describing the specified file. Normally, the *stat()* or *fstat()* routine is used to obtain file information, rather than using the *FIOFSTATGET* function directly. See the manual entry for *dirLib*:

```
struct stat statStruct;
fd = open ("file", O_RDONLY);
status = ioctl (fd, FIOFSTATGET, &statStruct);
```

**FIOFSTATFSGET**

Gets the file system parameters for an open file descriptor. The argument *statfsStruct* is a pointer to a *statfs* structure that is filled with data describing the underlying file system. Normally, the *stat()* or *fstat()* routine is used to obtain file information, rather than using the *FIOFSTATGET* function directly. See the manual entry for *dirLib*:

```
statfs statfsStruct;
fd = open ("directory", O_RDONLY);
status = ioctl (fd, FIOFSTATFSGET, &statfsStruct);
```

**DEFICIENCIES**

There is only one client handle/cache per task. Performance is poor if a task is accessing two or more NFS files.

Changing *nfsCacheSize* after a file is open could cause adverse effects. However, changing it before opening any NFS file descriptors should not pose a problem.

**INCLUDE FILES**

*nfsDrv.h*, *ioLib.h*, *dirent.h*

**SEE ALSO**

*dirLib*, *nfsLib*, *hostAdd()*, *ioctl()*,

---

## nfsLib

**NAME**

*nfsLib* – Network File System (NFS) library

**ROUTINES**

*nfsHelp()* - display the NFS help menu  
*nfsExportShow()* - display the exported file systems of a remote host  
*nfsAuthUnixPrompt()* - modify the NFS UNIX authentication parameters  
*nfsAuthUnixShow()* - display the NFS UNIX authentication parameters  
*nfsAuthUnixSet()* - set the NFS UNIX authentication parameters  
*nfsAuthUnixGet()* - get the NFS UNIX authentication parameters  
*nfsIdSet()* - set the ID number of the NFS UNIX authentication parameters

**ntPassFsLib**

<b>DESCRIPTION</b>	<p>This library provides the client side of services for NFS (Network File System) devices. Most routines in this library should not be called by users, but rather by device drivers. The driver is responsible for keeping track of file pointers, mounted disks, and cached buffers. This library uses Remote Procedure Calls (RPC) to make the NFS calls.</p> <p>VxWorks is delivered with NFS disabled. To use this feature, include the following component: <b>INCLUDE_NFS</b></p> <p>In the same file, <b>NFS_USER_ID</b> and <b>NFS_GROUP_ID</b> should be defined to set the default user ID and group ID at system start-up. For information about creating NFS devices, see the <i>WindNet TCP/IP Network Programmer's Guide</i>.</p> <p>Normal use of NFS requires no more than 2000 bytes of stack. This requirement may change depending on how the maximum file name path length parameter, <b>NFS_MAXPATH</b>, is configured. As many as 4 character arrays of length <b>NFS_MAXPATH</b> may be allocated off the stack during client operation. Therefore any increase in the parameter can increase stack usage by a factor of four times the deviation from default <b>NFS_MAXPATH</b>. For example, a change from 255 to 1024 will increase peak stack usage by <math>(1024 - 255) * 4</math> which is 3076 bytes.</p>
<b>NFS USER IDENTIFICATION</b>	<p>NFS is built on top of RPC and uses a type of RPC authentication known as <b>AUTH_UNIX</b>, which is passed on to the NFS server with every NFS request. <b>AUTH_UNIX</b> is a structure that contains necessary information for NFS, including the user ID number and a list of group IDs to which the user belongs. On UNIX systems, a user ID is specified in the file <b>/etc/passwd</b>. The list of groups to which a user belongs is specified in the file <b>/etc/group</b>.</p> <p>To change the default authentication parameters, use <b>nfsAuthUnixPrompt()</b>. To change just the <b>AUTH_UNIX</b> ID, use <b>nfsIdSet()</b>. Usually, only the user ID needs to be changed to indicate a new NFS user.</p>
<b>INCLUDE FILES</b>	<b>nfsLib.h</b>
<b>SEE ALSO</b>	<b>rpcLib</b> , <b>ioLib</b> , <b>nfsDrv</b>

---

## ntPassFsLib

<b>NAME</b>	<b>ntPassFsLib</b> – pass-through (to Windows NT) file system library
<b>ROUTINES</b>	<b>ntPassFsDevInit()</b> - associate a device with ntPassFs file system functions <b>ntPassFsInit()</b> - prepare to use the ntPassFs library
<b>DESCRIPTION</b>	This module is only used with VxSim simulated versions of VxWorks.

This library provides services for file-oriented device drivers to use the Windows NT file standard. In general, the routines in this library are not to be called directly by users, but rather by the VxWorks I/O System.

#### INITIALIZING PASSFSLIB

Before any other routines in **ntPassFsLib** can be used, the routine **ntPassFsInit()** must be called to initialize this library. The **ntPassFsDevInit()** routine associates a device name with the **ntPassFsLib** functions. The parameter expected by **ntPassFsDevInit()** is a pointer to a name string, to be used to identify the volume/device. This will be part of the pathname for I/O operations which operate on the device. This name will appear in the I/O system device table, which may be displayed using the **iosDevShow()** routine.

As an example:

```
ntPassFsInit (1);  
ntPassFsDevInit ("host:");
```

After the **ntPassFsDevInit()** call has been made, when **ntPassFsLib** receives a request from the I/O system, it calls the Windows NT I/O system to service the request. Only one volume may be created.

#### READING DIRECTORY ENTRIES

Directories on a **ntPassFs** volume may be searched using the **opendir()**, **readdir()**, **rewinddir()**, and **closedir()** routines. These calls allow the names of files and sub-directories to be determined.

To obtain more detailed information about a specific file, use the **fstat()** or **stat()** function. Along with standard file information, the structure used by these routines also returns the file attribute byte from a **ntPassFs** directory entry.

#### FILE DATE AND TIME

Windows NT file date and time are passed through to VxWorks.

**INCLUDE FILES**     **ntPassFsLib.h**

**SEE ALSO**         **ioLib, iosLib, dirLib, ramDrv**

---

## passFsLib

<b>NAME</b>	<b>passFsLib</b> – pass-through (to UNIX) file system library (VxSim)
<b>ROUTINES</b>	<b>passFsDevInit()</b> - associate a device with passFs file system functions <b>passFsInit()</b> - prepare to use the passFs library
<b>DESCRIPTION</b>	This module is only used with VxSim simulated versions of VxWorks.  This library provides services for file-oriented device drivers to use the UNIX file standard. This module takes care of all the buffering, directory maintenance, and file system details that are necessary. In general, the routines in this library are not to be called directly by users, but rather by the VxWorks I/O System.

### INITIALIZING PASSFSLIB

Before any other routines in **passFsLib** can be used, the routine **passFsInit()** must be called to initialize this library. The **passFsDevInit()** routine associates a device name with the **passFsLib** functions. The parameter expected by **passFsDevInit()** is a pointer to a name string, to be used to identify the volume/device. This will be part of the pathname for I/O operations which operate on the device. This name will appear in the I/O system device table, which may be displayed using the **iosDevShow()** routine.

As an example:

```
passFsInit (1);  
passFsDevInit ("host:");
```

After the **passFsDevInit()** call has been made, when **passFsLib** receives a request from the I/O system, it calls the UNIX I/O system to service the request. Only one volume may be created.

### READING DIRECTORY ENTRIES

Directories on a passFs volume may be searched using the **opendir()**, **readdir()**, **rewinddir()**, and **closedir()** routines. These calls allow the names of files and sub-directories to be determined.

To obtain more detailed information about a specific file, use the **fstat()** or **stat()** function. Along with standard file information, the structure used by these routines also returns the file attribute byte from a passFs directory entry.

### FILE DATE AND TIME

UNIX file date and time are passed though to VxWorks.

**INCLUDE FILES** **passFsLib.h**

**SEE ALSO** **ioLib, iosLib, dirLib, ramDrv**

---

## pentiumALib

<b>NAME</b>	<b>pentiumALib</b> – Pentium and PentiumPro specific routines
<b>ROUTINES</b>	<p><b>pentiumCr4Get()</b> - get contents of CR4 register <b>pentiumCr4Set()</b> - sets specified value to the CR4 register <b>pentiumP6PmcStart()</b> - start both PMC0 and PMC1 <b>pentiumP6PmcStop()</b> - stop both PMC0 and PMC1 <b>pentiumP6PmcStop1()</b> - stop PMC1 <b>pentiumP6PmcGet()</b> - get the contents of PMC0 and PMC1 <b>pentiumP6PmcGet0()</b> - get the contents of PMC0 <b>pentiumP6PmcGet1()</b> - get the contents of PMC1 <b>pentiumP6PmcReset()</b> - reset both PMC0 and PMC1 <b>pentiumP6PmcReset0()</b> - reset PMC0 <b>pentiumP6PmcReset1()</b> - reset PMC1 <b>pentiumP5PmcStart0()</b> - start PMC0 <b>pentiumP5PmcStart1()</b> - start PMC1 <b>pentiumP5PmcStop()</b> - stop both P5 PMC0 and PMC1 <b>pentiumP5PmcStop0()</b> - stop P5 PMC0 <b>pentiumP5PmcStop1()</b> - stop P5 PMC1 <b>pentiumP5PmcGet()</b> - get the contents of P5 PMC0 and PMC1 <b>pentiumP5PmcGet0()</b> - get the contents of P5 PMC0 <b>pentiumP5PmcGet1()</b> - get the contents of P5 PMC1 <b>pentiumP5PmcReset()</b> - reset both PMC0 and PMC1 <b>pentiumP5PmcReset0()</b> - reset PMC0 <b>pentiumP5PmcReset1()</b> - reset PMC1 <b>pentiumTscGet64()</b> - get 64Bit TSC (Timestamp Counter) <b>pentiumTscGet32()</b> - get the lower half of the 64Bit TSC (Timestamp Counter) <b>pentiumTscReset()</b> - reset the TSC (Timestamp Counter) <b>pentiumMsrGet()</b> - get the contents of the specified MSR (Model Specific Register) <b>pentiumMsrSet()</b> - set a value to the specified MSR (Model Specific Registers) <b>pentiumTlbFlush()</b> - flush TLBs (Translation Lookaside Buffers) <b>pentiumSerialize()</b> - execute a serializing instruction CPUID <b>pentiumBts()</b> - execute atomic compare-and-exchange instruction to set a bit <b>pentiumBtc()</b> - execute atomic compare-and-exchange instruction to clear a bit</p>
<b>DESCRIPTION</b>	This module contains Pentium and PentiumPro specific routines written in assembly language.

### MCA (Machine Check Architecture)

The Pentium processor introduced a new exception called the machine-check exception (interrupt-18). This exception is used to signal hardware-related errors, such as a parity error on a read cycle. The PentiumPro processor extends the types of errors that can be detected and that generate a machine-check exception. It also provides a new

machine-check architecture that records information about a machine-check error and provides the basis for an extended error logging capability.

MCA is enabled and its status registers are cleared zero in `sysHwInit()`. Its registers are accessed by `pentiumMsrSet()` and `pentiumMsrGet()`.

#### PMC (Performance Monitoring Counters)

The P5 and P6 family of processor has two performance-monitoring counters for use in monitoring internal hardware operations. These counters are duration or event counters that can be programmed to count any of approximately 100 different types of events, such as the number of instructions decoded, number of interrupts received, or number of cache loads. However, the set of events can be counted with PMC is different in the P5 and P6 family of processors; and the locations and bit definitions of the related counter and control registers are also different. So there are two set of PMC routines, one for P6 family and one for p5 family respectively.

There are nine routines to interface the PMC of P6 family processors. These nine routines are:

```
STATUS pentiumP6PmcStart
(
    int pmcEvtSel0;      /* performance event select register 0 */
    int pmcEvtSel1;      /* performance event select register 1 */
)

void pentiumP6PmcStop (void)
void pentiumP6PmcStop1 (void)
void pentiumP6PmcGet
(
    long long int * pPmc0; /* performance monitoring counter 0 */
    long long int * pPmc1; /* performance monitoring counter 1 */
)

void pentiumP6PmcGet0
(
    long long int * pPmc0; /* performance monitoring counter 0 */
)

void pentiumP6PmcGet1
(
    long long int * pPmc1; /* performance monitoring counter 1 */
)

void pentiumP6PmcReset (void)
void pentiumP6PmcReset0 (void)
void pentiumP6PmcReset1 (void)
```

`pentiumP6PmcStart()` starts both PMC0 and PMC1. `pentiumP6PmcStop()` stops them, and `pentiumP6PmcStop1()` stops only PMC1. `pentiumP6PmcGet()` gets contents of PMC0 and PMC1. `pentiumP6PmcGet0()` gets contents of PMC0, and `pentiumP6PmcGet1()` gets contents of PMC1. `pentiumP6PmcReset()` resets both PMC0

and PMC1. **pentiumP6PmcReset0()** resets PMC0, and **pentiumP6PmcReset1()** resets PMC1. PMC is enabled in **sysHwInit()**. Selected events in the default configuration are PMC0 = number of hardware interrupts received and PMC1 = number of misaligned data memory references.

There are ten routines to interface the PMC of P5 family processors. These ten routines are:

```

STATUS pentiumP5PmcStart0
(
    int pmc0Cesr; /* PMC0 control and event select */
)
STATUS pentiumP5PmcStart1
(
    int pmc1Cesr; /* PMC1 control and event select */
)
void pentiumP5PmcStop0 (void)
void pentiumP5PmcStop1 (void)
void pentiumP5PmcGet
(
    long long int * pPmc0; /* performance monitoring counter 0 */
    long long int * pPmc1; /* performance monitoring counter 1 */
)
void pentiumP5PmcGet0
(
    long long int * pPmc0; /* performance monitoring counter 0 */
)
void pentiumP5PmcGet1
(
    long long int * pPmc1; /* performance monitoring counter 1 */
)
void pentiumP5PmcReset (void)
void pentiumP5PmcReset0 (void)
void pentiumP5PmcReset1 (void)

```

**pentiumP5PmcStart0()** starts PMC0, and **pentiumP5PmcStart1()** starts PMC1. **pentiumP5PmcStop0()** stops PMC0, and **pentiumP5PmcStop1()** stops PMC1. **pentiumP5PmcGet()** gets contents of PMC0 and PMC1. **pentiumP5PmcGet0()** gets contents of PMC0, and **pentiumP5PmcGet1()** gets contents of PMC1. **pentiumP5PmcReset()** resets both PMC0 and PMC1. **pentiumP5PmcReset0()** resets PMC0, and **pentiumP5PmcReset1()** resets PMC1. PMC is enabled in **sysHwInit()**. Selected events in the default configuration are PMC0 = number of hardware interrupts received and PMC1 = number of misaligned data memory references.

#### MSR (Model Specific Register)

The concept of model-specific registers (MSRs) to control hardware functions in the processor or to monitor processor activity was introduced in the PentiumPro processor.

The new registers control the debug extensions, the performance counters, the machine-check exception capability, the machine check architecture, and the MTRRs. The MSRs can be read and written to using the RDMSR and WRMSR instructions, respectively.

There are two routines to interface the MSR. These two routines are:

```
void pentiumMsrGet
(
    int address,          /* MSR address */
    long long int * pData /* MSR data */
)
void pentiumMsrSet
(
    int address,          /* MSR address */
    long long int * pData /* MSR data */
)
```

**pentiumMsrGet()** get contents of the specified MSR, and **pentiumMsrSet()** sets value to the specified MSR.

#### TSC (Time Stamp Counter)

The PentiumPro processor provides a 64-bit time-stamp counter that is incremented every processor clock cycle. The counter is incremented even when the processor is halted by the HLT instruction or the external STPCLK# pin. The time-stamp counter is set to 0 following a hardware reset of the processor. The RDTSC instruction reads the time stamp counter and is guaranteed to return a monotonically increasing unique value whenever executed, except for 64-bit counter wraparound. Intel guarantees, architecturally, that the time-stamp counter frequency and configuration will be such that it will not wraparound within 10 years after being reset to 0. The period for counter wrap is several thousands of years in the PentiumPro and Pentium processors.

There are three routines to interface the TSC. These three routines are:

```
void pentiumTscReset (void)
void pentiumTscGet32 (void)
void pentiumTscGet64
(
    long long int * pTsc /* TSC */
)
```

**pentiumTscReset()** resets the TSC. **pentiumTscGet32()** gets the lower half of the 64Bit TSC, and **pentiumTscGet64()** gets the entire 64Bit TSC.

Four other routines are provided in this library. They are:

```
void pentiumFlbFlush (void)
void pentiumSerialize (void)
STATUS pentiumBts
```

```

(
char * pFlag          /* flag address */
)
STATUS pentiumBtc (pFlag)
(
char * pFlag          /* flag address */
)

```

**pentiumTlbFlush()** flushes TLBs (Translation Lookaside Buffers). **pentiumSerialize()** does serialization by executing CPUID instruction. **pentiumBts()** executes an atomic compare-and-exchange instruction to set a bit. **pentiumBtc()** executes an atomic compare-and-exchange instruction to clear a bit.

**SEE ALSO** *Pentium, PentiumPro Family Developer's Manual*

---

## pentiumLib

**NAME** **pentiumLib** – Pentium and Pentium[234] library

**ROUTINES**

- pentiumMtrrEnable()** - enable MTRR (Memory Type Range Register)
- pentiumMtrrDisable()** - disable MTRR (Memory Type Range Register)
- pentiumMtrrGet()** - get MTRRs to a specified MTRR table
- pentiumMtrrSet()** - set MTRRs from specified MTRR table with WRMSR instruction.
- pentiumPmcStart()** - start both PMC0 and PMC1
- pentiumPmcStart0()** - start PMC0
- pentiumPmcStart1()** - start PMC1
- pentiumPmcStop()** - stop both PMC0 and PMC1
- pentiumPmcStop0()** - stop PMC0
- pentiumPmcStop1()** - stop PMC1
- pentiumPmcGet()** - get the contents of PMC0 and PMC1
- pentiumPmcGet0()** - get the contents of PMC0
- pentiumPmcGet1()** - get the contents of PMC1
- pentiumPmcReset()** - reset both PMC0 and PMC1
- pentiumPmcReset0()** - reset PMC0
- pentiumPmcReset1()** - reset PMC1
- pentiumMsrInit()** - initialize all the MSRs (Model Specific Register)
- pentiumMcaEnable()** - enable/disable the MCA (Machine Check Architecture)

**DESCRIPTION** This library provides Pentium and Pentium[234] specific routines.

**MTRR (Memory Type Range Register)**

MTRR (Memory Type Range Register) are a new feature introduced in the P6 family

processor that allow the processor to optimize memory operations for different types of memory, such as RAM, ROM, frame buffer memory, and memory-mapped IO. MTRRs configure an internal map of how physical address ranges are mapped to various types of memory. The processor uses this internal map to determine the cacheability of various physical memory locations and the optimal method of accessing memory locations. For example, if a memory location is specified in an MTRR as write-through memory, the processor handles accesses to this location as follows. It reads data from that location in lines and caches the read data or maps all writes to that location to the bus and updates the cache to maintain cache coherency. In mapping the physical address space with MTRRs, the processor recognizes five types of memory: uncacheable (UC), write-combining (WC), write-through (WT), write-protected (WP), and write-back (WB).

There is one table - `sysMtrr[]` in `sysLib.c` - and four routines to interface the MTRR. These four routines are:

```
void pentiumMtrrEnable (void)
void pentiumMtrrDisable (void)
STATUS pentiumMtrrGet
(
    MTRR * pMtrr          /* MTRR table */
)
STATUS pentiumMtrrSet (void)
(
    MTRR * pMtrr          /* MTRR table */
)
```

`pentiumMtrrEnable()` enables MTRR, `pentiumMtrrDisable()` disables MTRR. `pentiumMtrrGet()` gets MTRRs to the specified MTRR table. `pentiumMtrrSet()` sets MTRRs from the specified MTRR table. The MTRR table is defined as follows:

```
typedef struct mtrr_fix          /* MTRR - fixed range register */
{
    char type[8];                /* address range: [0]=0-7 ... [7]=56-63 */
} MTRR_FIX;
typedef struct mtrr_var          /* MTRR - variable range register */
{
    long long int base;          /* base register */
    long long int mask;         /* mask register */
} MTRR_VAR;
typedef struct mtrr              /* MTRR */
{
    int cap[2];                  /* MTRR cap register */
    int defType[2];              /* MTRR defType register */
    MTRR_FIX fix[11];           /* MTRR fixed range registers */
    MTRR_VAR var[8];            /* MTRR variable range registers */
} MTRR;
```

Fixed Range Register's type array can be one of following memory types. **MTRR\_UC** (uncacheable), **MTRR\_WC** (write-combining), **MTRR\_WT** (write-through), **MTRR\_WP** (write-protected), and **MTRR\_WB** (write-back). MTRR is enabled in **sysHwInit( )**.

#### PMC (Performance Monitoring Counters)

The P5 and P6 family of processors has two performance-monitoring counters for use in monitoring internal hardware operations. These counters are duration or event counters that can be programmed to count any of approximately 100 different types of events, such as the number of instructions decoded, number of interrupts received, or number of cache loads. However, the set of events can be counted with PMC is different in the P5 and P6 family of processors; and the locations and bit definitions of the related counter and control registers are also different. So there are two set of PMC routines, one for P6 family and one for P5 family respectively in **pentiumALib**. For convenience, the PMC routines here are acting as wrappers to those routines in **pentiumALib**. They will call the P5 or P6 routine depending on the processor type.

There are twelve routines to interface the PMC. These twelve routines are:

```

STATUS pentiumPmcStart
(
    int pmcEvtSel0;          /* performance event select register 0 */
    int pmcEvtSel1;          /* performance event select register 1 */
)

STATUS pentiumPmcStart0
(
    int pmcEvtSel0;          /* performance event select register 0 */
)

STATUS pentiumPmcStart1
(
    int pmcEvtSel1;          /* performance event select register 1 */
)

void pentiumPmcStop (void)
void pentiumPmcStop0 (void)
void pentiumPmcStop1 (void)
void pentiumPmcGet
(
    long long int * pPmc0; /* performance monitoring counter 0 */
    long long int * pPmc1; /* performance monitoring counter 1 */
)

void pentiumPmcGet0
(
    long long int * pPmc0; /* performance monitoring counter 0 */
)

void pentiumPmcGet1
(
    long long int * pPmc1; /* performance monitoring counter 1 */

```

```
    )  
void pentiumPmcReset (void)  
void pentiumPmcReset0 (void)  
void pentiumPmcReset1 (void)
```

**pentiumPmcStart()** starts both PMC0 and PMC1. **pentiumPmcStart0()** starts PMC0, and **pentiumPmcStart1()** starts PMC1. **pentiumPmcStop()** stops both PMC0 and PMC1. **pentiumPmcStop0()** stops PMC0, and **pentiumPmcStop1()** stops PMC1. **pentiumPmcGet()** gets contents of PMC0 and PMC1. **pentiumPmcGet0()** gets contents of PMC0, and **pentiumPmcGet1()** gets contents of PMC1. **pentiumPmcReset()** resets both PMC0 and PMC1. **pentiumPmcReset0()** resets PMC0, and **pentiumPmcReset1()** resets PMC1. PMC is enabled in **sysHwInit()**. Selected events in the default configuration are PMC0 = number of hardware interrupts received and PMC1 = number of misaligned data memory references.

#### **MSR (Model Specific Registers)**

The P5(Pentium), P6(PentiumPro, II, III), and P7(Pentium4) family processors contain a model-specific registers (MSRs). These registers are implementation specific. They are provided to control a variety of hardware and software related features including the performance monitoring, the debug extensions, the machine check architecture, *etc.*

There is one routine - **pentiumMsrInit()** - to initialize all the MSRs. This routine initializes all the MSRs in the processor and works on either P5, P6 or P7 family processors.

#### **MCA (Machine Check Architecture)**

The P5(Pentium), P6(PentiumPro, II, III), and P7(Pentium4) family processors have a machine-check architecture that provides a mechanism for detecting and reporting hardware (machine) errors, such as system bus errors, ECC errors, parity errors, cache errors and TLB errors. It consists of a set of model-specific registers (MSRs) that are used to set up machine checking and additional banks of MSRs for recording errors that are detected. The processor signals the detection of a machine-check error by generating a machine-check exception, which an abort class exception. The implementation of the machine-check architecture, does not ordinarily permit the processor to be restarted reliably after generating a machine-check exception. However, the machine-check exception handler can collect information about the machine-check error from the machine-check MSRs.

There is one routine - **pentiumMcaEnable()** - to enable or disable the MCA. The routine enables or disables 1) the Machine Check Architecture and its Error Reporting register banks 2) the Machine Check Exception by toggling the MCE bit in the CR4. This routine works on either P5, P6 or P7 family.

#### **SEE ALSO**

**PentiumALib**, *Pentium*, *Pentium[234] Family Developer's Manual*

---

## pentiumShow

<b>NAME</b>	<b>pentiumShow</b> – Pentium and Pentium[234] specific show routines
<b>ROUTINES</b>	<b>pentiumMcaShow()</b> - show MCA (Machine Check Architecture) registers <b>pentiumPmcShow()</b> - show PMCs (Performance Monitoring Counters) <b>pentiumMsrShow()</b> - show all the MSR (Model Specific Register)
<b>DESCRIPTION</b>	This library provides Pentium and Pentium[234] specific show routines. <b>pentiumMcaShow()</b> shows Machine Check Global Control Registers and Error Reporting Register Banks. <b>pentiumPmcShow()</b> shows PMC0 and PMC1, and reset them if the parameter zap is <b>TRUE</b> .
<b>SEE ALSO</b>	<i>VxWorks Programmer's Guide: Configuration</i>

---

## pingLib

<b>NAME</b>	<b>pingLib</b> – Packet InterNet Groper (PING) library
<b>ROUTINES</b>	<b>pingLibInit()</b> - initialize the <b>ping()</b> utility <b>ping()</b> - test that a remote host is reachable
<b>DESCRIPTION</b>	This library contains the <b>ping()</b> utility, which tests the reachability of a remote host. The routine <b>ping()</b> is typically called from the VxWorks shell to check the network connection to another VxWorks target or to a UNIX host. <b>ping()</b> may also be used programmatically by applications that require such a test. The remote host must be running TCP/IP networking code that responds to ICMP echo request packets. The <b>ping()</b> routine is re-entrant, thus may be called by many tasks concurrently. The routine <b>pingLibInit()</b> initializes the <b>ping()</b> utility and allocates resources used by this library. It is called automatically when <b>INCLUDE_PING</b> is defined.

## pipeDrv

<b>NAME</b>	<b>pipeDrv</b> – pipe I/O driver
<b>ROUTINES</b>	<b>pipeDrv()</b> - initialize the pipe driver <b>pipeDevCreate()</b> - create a pipe device <b>pipeDevDelete()</b> - delete a pipe device
<b>DESCRIPTION</b>	<p>The pipe driver provides a mechanism that lets tasks communicate with each other through the standard I/O interface. Pipes can be read and written with normal <b>read()</b> and <b>write()</b> calls. The pipe driver is initialized with <b>pipeDrv()</b>. Pipe devices are created with <b>pipeDevCreate()</b>.</p> <p>The pipe driver uses the VxWorks message queue facility to do the actual buffering and delivering of messages. The pipe driver simply provides access to the message queue facility through the I/O system. The main differences between using pipes and using message queues directly are:</p> <ul style="list-style-type: none"><li>– pipes are named (with I/O device names).</li><li>– pipes use the standard I/O functions -- <b>open()</b>, <b>close()</b>, <b>read()</b>, <b>write()</b> -- while message queues use the functions <b>msgQSend()</b> and <b>msgQReceive()</b>.</li><li>– pipes respond to standard <b>ioctl()</b> functions.</li><li>– pipes can be used in a <b>select()</b> call.</li><li>– message queues have more flexible options for timeouts and message priorities.</li><li>– pipes are less efficient than message queues because of the additional overhead of the I/O system.</li></ul>
<b>INSTALLING THE DRIVER</b>	<p>Before using the driver, it must be initialized and installed by calling <b>pipeDrv()</b>. This routine must be called before any pipes are created. It is called automatically by the root task, <b>usrRoot()</b>, in <b>usrConfig.c</b> when the configuration macro <b>INCLUDE_PIPES</b> is defined.</p>
<b>CREATING PIPES</b>	<p>Before a pipe can be used, it must be created with <b>pipeDevCreate()</b>. For example, to create a device pipe <b>/pipe/demo</b> with up to 10 messages of size 100 bytes, the proper call is:</p> <pre>pipeDevCreate ("/pipe/demo", 10, 100);</pre>
<b>USING PIPES</b>	<p>Once a pipe has been created it can be opened, closed, read, and written just like any other I/O device. Often the data that is read and written to a pipe is a structure of some type. Thus, the following example writes to a pipe and reads back the same data:</p>

```

{
int fd;
struct msg outMsg;
struct msg inMsg;
int len;
fd = open ("/pipe/demo", O_RDWR);
write (fd, &outMsg, sizeof (struct msg));
len = read (fd, &inMsg, sizeof (struct msg));
close (fd);
}

```

The data written to a pipe is kept as a single message and will be read all at once in a single read. If `read()` is called with a buffer that is smaller than the message being read, the remainder of the message will be discarded. Thus, pipe I/O is “message oriented” rather than “stream oriented.” In this respect, VxWorks pipes differ significantly from UNIX pipes which are stream oriented and do not preserve message boundaries.

#### WRITING TO PIPES FROM INTERRUPT SERVICE ROUTINES

Interrupt service routines (ISR) can write to pipes, providing one of several ways in which ISRs can communicate with tasks. For example, an interrupt service routine may handle the time-critical interrupt response and then send a message on a pipe to a task that will continue with the less critical aspects. However, the use of pipes to communicate from an ISR to a task is now discouraged in favor of the direct message queue facility, which offers lower overhead (see the manual entry for `msgQLib` for more information).

#### SELECT CALLS

An important feature of pipes is their ability to be used in a `select()` call. The `select()` routine allows a task to wait for input from any of a selected set of I/O devices. A task can use `select()` to wait for input from any combination of pipes, sockets, or serial devices. See the manual entry for `select()`.

#### IOCTL FUNCTIONS

Pipe devices respond to the following `ioctl()` functions. These functions are defined in the header file `ioLib.h`.

##### FIOGETNAME

Gets the file name of `fd` and copies it to the buffer referenced by `nameBuf`:

```
status = ioctl (fd, FIOGETNAME, &nameBuf);
```

##### FIONREAD

Copies to `nBytesUnread` the number of bytes remaining in the first message in the pipe:

```
status = ioctl (fd, FIONREAD, &nBytesUnread);
```

##### FIONMSGS

Copies to `nMessages` the number of discrete messages remaining in the pipe:

```
status = ioctl (fd, FIONMSGS, &nMessages);
```

**pppHookLib**

**FIOFLUSH**

Discards all messages in the pipe and releases the memory block that contained them:

```
status = ioctl (fd, FIOFLUSH, 0);
```

**INCLUDE FILES** ioLib.h, pipeDrv.h

**SEE ALSO** select(), msgQLib, *VxWorks Programmer's Guide: I/O System*

---

## pppHookLib

**NAME** pppHookLib – PPP hook library

**ROUTINES** pppHookAdd() - add a hook routine on a unit basis  
pppHookDelete() - delete a hook routine on a unit basis

**DESCRIPTION** This library provides routines to add and delete connect and disconnect routines. The connect routine, added on a unit basis, is called before the initial phase of link option negotiation. The disconnect routine, added on a unit basis is called before the PPP connection is closed. These connect and disconnect routines can be used to hook up additional software. If either connect or disconnect hook returns **ERROR**, the connection is terminated immediately.

This library is automatically linked into the VxWorks system image when the configuration macro **INCLUDE\_PPP** is defined.

**INCLUDE FILES** pppLib.h

**SEE ALSO** pppLib, *VxWorks Programmer's Guide: Network*

---

## pppLib

**NAME** pppLib – Point-to-Point Protocol library

**ROUTINES** pppInit() - initialize a PPP network interface  
pppDelete() - delete a PPP network interface

**DESCRIPTION** This library implements the VxWorks Point-to-Point Protocol (PPP) facility. PPP allows VxWorks to communicate with other machines by sending encapsulated multi-protocol datagrams over a point-to-point serial link. VxWorks may have up to 16 PPP interfaces

active at any one time. Each individual interface (or “unit”) operates independent of the state of other PPP units.

#### USER-CALLABLE ROUTINES

PPP network interfaces are initialized using the **pppInit()** routine. This routine’s parameters specify the unit number, the name of the serial interface (*tty*) device, Internet (IP) addresses for both ends of the link, the interface baud rate, an optional pointer to a configuration options structure, and an optional pointer to a configuration options file. The **pppDelete()** routine deletes a specified PPP interface.

#### DATA ENCAPSULATION

PPP uses HDLC-like framing, in which five header and three trailer octets are used to encapsulate each datagram. In environments where bandwidth is at a premium, the total encapsulation may be shortened to four octets with the available address/control and protocol field compression options.

#### LINK CONTROL PROTOCOL

PPP incorporates a link-layer protocol called Link Control Protocol (LCP), which is responsible for the link set up, configuration, and termination. LCP provides for automatic negotiation of several link options, including datagram encapsulation format, user authentication, and link monitoring (LCP echo request/reply).

#### NETWORK CONTROL PROTOCOLS

PPP’s Network Control Protocols (NCP) allow PPP to support different network protocols. VxWorks supports only one NCP, the Internet Protocol Control Protocol (IPCP), which allows the establishment and configuration of IP over PPP links. IPCP supports the negotiation of IP addresses and TCP/IP header compression (commonly called “VJ” compression).

**AUTHENTICATION** The VxWorks PPP implementation supports two separate user authentication protocols: the Password Authentication Protocol (PAP) and the Challenge-Handshake Authentication Protocol (CHAP). While PAP only authenticates at the time of link establishment, CHAP may be configured to periodically require authentication throughout the life of the link. Both protocols are independent of one another, and either may be configured in through the PPP options structure or options file.

**IMPLEMENTATION** Each VxWorks PPP interface is handled by two tasks: the daemon task (*tPPPunit*) and the write task (*tPPPunitWrt*).

The daemon task controls the various PPP control protocols (LCP, IPCP, CHAP, and PAP). Each PPP interface has its own daemon task that handles link set up, negotiation of link options, link-layer user authentication, and link termination. The daemon task is not used for the actual sending and receiving of IP datagrams.

The write task controls the transmit end of a PPP driver interface. Each PPP interface has its own write task that handles the actual sending of a packet by writing data to the *tty*

device. Whenever a packet is ready to be sent out, the PPP driver activates this task by giving a semaphore. The write task then completes the packet framing and writes the packet data to the *tty* device.

The receive end of the PPP interface is implemented as a “hook” into the *tty* device driver. The *tty* driver’s receive interrupt service routine (ISR) calls the PPP driver’s ISR every time a character is received on the serial channel. When the correct PPP framing character sequence is received, the PPP ISR schedules the **tNetTask** task to call the PPP input routine. The PPP input routine reads a whole PPP packet out of the *tty* ring buffer and processes it according to PPP framing rules. The packet is then queued either to the IP input queue or to the PPP daemon task input queue.

**INCLUDE FILES**     **pppLib.h**

**SEE ALSO**            **ifLib**, **tyLib**, **pppSecretLib**, **pppShow**, *VxWorks Programmer’s Guide: Network, RFC-1332: The PPP Internet Protocol Control Protocol (IPCP), RFC-1334: PPP Authentication Protocols, RFC-1548: The Point-to-Point Protocol (PPP), RFC-1549: PPP in HDLC Framing*

**ACKNOWLEDGEMENT**

This program is based on original work done by Paul Mackerras of Australian National University, Brad Parker, Greg Christy, Drew D. Perkins, Rick Adams, and Chris Torek.

---

## pppSecretLib

**NAME**                **pppSecretLib** – PPP authentication secrets library

**ROUTINES**           **pppSecretAdd()** - add a secret to the PPP authentication secrets table  
**pppSecretDelete()** - delete a secret from the PPP authentication secrets table

**DESCRIPTION**        This library provides routines to create and manipulate a table of “secrets” for use with Point-to-Point Protocol (PPP) user authentication protocols. The secrets in the secrets table can be searched by peers on a PPP link so that one peer (client) can send a secret word to the other peer (server). If the client cannot find a suitable secret when required to do so, or the secret received by the server is not valid, the PPP link may be terminated.

This library is automatically linked into the VxWorks system image when the configuration macro **INCLUDE\_PPP** is defined.

**INCLUDE FILES**     **pppLib.h**

**SEE ALSO**            **pppLib**, **pppShow**, *VxWorks Programmer’s Guide: Network*

---

## pppShow

<b>NAME</b>	<b>pppShow</b> – Point-to-Point Protocol show routines
<b>ROUTINES</b>	<b>pppInfoShow()</b> - display PPP link status information <b>pppInfoGet()</b> - get PPP link status information <b>pppstatShow()</b> - display PPP link statistics <b>pppstatGet()</b> - get PPP link statistics <b>pppSecretShow()</b> - display the PPP authentication secrets table
<b>DESCRIPTION</b>	<p>This library provides routines to show Point-to-Point Protocol (PPP) link status information and statistics. Also provided are routines that programmatically access this same information.</p> <p>This library is automatically linked into the VxWorks system image when the configuration macro <b>INCLUDE_PPP</b> is defined.</p>
<b>INCLUDE FILES</b>	<b>pppLib.h</b>
<b>SEE ALSO</b>	<b>pppLib</b> , <i>VxWorks Programmer's Guide: Network</i>

---

## proxyArpLib

<b>NAME</b>	<b>proxyArpLib</b> – proxy Address Resolution Protocol (ARP) server library
<b>ROUTINES</b>	<b>proxyArpLibInit()</b> - initialize proxy ARP <b>proxyNetCreate()</b> - create a proxy ARP network <b>proxyNetDelete()</b> - delete a proxy network <b>proxyNetShow()</b> - show proxy ARP networks <b>proxyPortFwdOn()</b> - enable broadcast forwarding for a particular port <b>proxyPortFwdOff()</b> - disable broadcast forwarding for a particular port <b>proxyPortShow()</b> - show ports enabled for broadcast forwarding
<b>DESCRIPTION</b>	<p>This library implements a proxy ARP server that uses the Address Resolution Protocol (ARP) to make physically distinct networks appear as one logical network (that is, the networks share the same address space). The server forwards ARP messages between the separate networks so that hosts on the main network can access hosts on the proxy network without altering their routing tables.</p> <p>The <b>proxyArpLibInit()</b> initializes the server and adds this library to the VxWorks image. This happens automatically if <b>INCLUDE_PROXY_SERVER</b> is defined at the time the image</p>

is built. The **proxyNetCreate()** and **proxyNetDelete()** routines will enable and disable the forwarding of ARP messages between networks. The **proxyNetShow()** routine displays the current set of proxy networks and the main network and known clients for each.

By default, this server automatically adds a client when it first detects an ARP message from that host. A VxWorks target can also register as a client with the **proxyReg()** routine and remove that registration with the **proxyUnreg()** routine. See the **proxyLib** manual pages for details.

To minimize traffic on the main network, the proxy server will only forward broadcast packets to the specified destination ports visible with the **proxyPortShow()** routine. The **proxyPortFwdOn()** and **proxyPortFwdOff()** routines will alter the current settings. Initially, broadcast forwarding is not active for any ports.

#### VXWORKS AE PROTECTION DOMAINS

Under VxWorks AE, the functions you assign for either **proxyArpHook** or **proxyBroadcastHook** must be valid within the kernel protection domain. This restriction does not apply under non-AE versions of VxWorks.

**INCLUDE FILES**     **proxyArpLib.h**

**SEE ALSO**            **proxyLib**, RFC 925, RFC 1027, RFC 826

---

## proxyLib

**NAME**                **proxyLib** – proxy Address Resolution Protocol (ARP) client library

**ROUTINES**          **proxyReg()** - register a proxy client  
**proxyUnreg()** - unregister a proxy client

**DESCRIPTION**      This library implements the client side of the proxy Address Resolution Protocol (ARP). It allows a VxWorks target to register itself as a proxy client by calling **proxyReg()** and to unregister itself by calling **proxyUnreg()**.

Both commands take an interface name and an IP address as arguments. The interface, *ifName*, specifies the interface through which to send the message. *ifName* must be a backplane interface. *proxyAddr* is the IP address associated with the interface *ifName*.

To use this feature, include **INCLUDE\_PROXY\_CLIENT**.

**INCLUDE FILES**     **proxyArpLib.h**

**SEE ALSO**            **proxyArpLib**

---

# pthreadLib

<b>NAME</b>	<b>pthreadLib</b> – POSIX 1003.1c thread library interfaces
<b>ROUTINES</b>	<p><b>pthreadLibInit()</b> - initialize POSIX threads support</p> <p><b>pthread_sigmask()</b> - change and/or examine calling thread's signal mask (POSIX)</p> <p><b>pthread_kill()</b> - send a signal to a thread (POSIX)</p> <p><b>pthread_mutexattr_init()</b> - initialize mutex attributes object (POSIX)</p> <p><b>pthread_mutexattr_destroy()</b> - destroy mutex attributes object (POSIX)</p> <p><b>pthread_mutexattr_setprotocol()</b> - set protocol attribute in mutex attributes object (POSIX)</p> <p><b>pthread_mutexattr_getprotocol()</b> - get value of protocol in mutex attributes object (POSIX)</p> <p><b>pthread_mutexattr_setprioceiling()</b> - set <i>prioceiling</i> attribute in mutex attributes object (POSIX)</p> <p><b>pthread_mutexattr_getprioceiling()</b> - get the current value of the <i>prioceiling</i> attribute in a mutex attributes object (POSIX)</p> <p><b>pthread_mutex_getprioceiling()</b> - get the value of the <i>prioceiling</i> attribute of a mutex (POSIX)</p> <p><b>pthread_mutex_setprioceiling()</b> - dynamically set the <i>prioceiling</i> attribute of a mutex (POSIX)</p> <p><b>pthread_mutex_init()</b> - initialize mutex from attributes object (POSIX)</p> <p><b>pthread_mutex_destroy()</b> - destroy a mutex (POSIX)</p> <p><b>pthread_mutex_lock()</b> - lock a mutex (POSIX)</p> <p><b>pthread_mutex_trylock()</b> - lock mutex if it is available (POSIX)</p> <p><b>pthread_mutex_unlock()</b> - unlock a mutex (POSIX)</p> <p><b>pthread_condattr_init()</b> - initialize a condition attribute object (POSIX)</p> <p><b>pthread_condattr_destroy()</b> - destroy a condition attributes object (POSIX)</p> <p><b>pthread_cond_init()</b> - initialize condition variable (POSIX)</p> <p><b>pthread_cond_destroy()</b> - destroy a condition variable (POSIX)</p> <p><b>pthread_cond_signal()</b> - unblock a thread waiting on a condition (POSIX)</p> <p><b>pthread_cond_broadcast()</b> - unblock all threads waiting on a condition (POSIX)</p> <p><b>pthread_cond_wait()</b> - wait for a condition variable (POSIX)</p> <p><b>pthread_cond_timedwait()</b> - wait for a condition variable with a timeout (POSIX)</p> <p><b>pthread_attr_setscope()</b> - set contention scope for thread attributes (POSIX)</p> <p><b>pthread_attr_getscope()</b> - get contention scope from thread attributes (POSIX)</p> <p><b>pthread_attr_setinheritsched()</b> - set <i>inheritsched</i> attribute in thread attribute object (POSIX)</p> <p><b>pthread_attr_getinheritsched()</b> - get current value if <i>inheritsched</i> attribute in thread attributes object (POSIX)</p> <p><b>pthread_attr_setschedpolicy()</b> - set <i>schedpolicy</i> attribute in thread attributes object (POSIX)</p> <p><b>pthread_attr_getschedpolicy()</b> - get <i>schedpolicy</i> attribute from thread attributes object (POSIX)</p>

**pthreadLib**

**pthread\_attr\_setschedparam()** - set *schedparam* attribute in thread attributes object (POSIX)

**pthread\_attr\_getschedparam()** - get value of *schedparam* attribute from thread attributes object (POSIX)

**pthread\_getschedparam()** - get value of *schedparam* attribute from a thread (POSIX)

**pthread\_setschedparam()** - dynamically set *schedparam* attribute for a thread (POSIX)

**pthread\_attr\_init()** - initialize thread attributes object (POSIX)

**pthread\_attr\_destroy()** - destroy a thread attributes object (POSIX)

**pthread\_attr\_setname()** - set name in thread attribute object

**pthread\_attr\_getname()** - get name of thread attribute object

**pthread\_attr\_setstacksize()** - set *stacksize* attribute in thread attributes

**pthread\_attr\_getstacksize()** - get stack value of *stacksize* attribute from thread attributes object (POSIX)

**pthread\_attr\_setstackaddr()** - set *stackaddr* attribute in thread attributes object (POSIX)

**pthread\_attr\_getstackaddr()** - get value of *stackaddr* attribute from thread attributes object (POSIX)

**pthread\_attr\_setdetachstate()** - set *detachstate* attribute in thread attributes object (POSIX)

**pthread\_attr\_getdetachstate()** - get value of *detachstate* attribute from thread attributes object (POSIX)

**pthread\_create()** - create a thread (POSIX)

**pthread\_detach()** - dynamically detach a thread (POSIX)

**pthread\_join()** - wait for a thread to terminate (POSIX)

**pthread\_exit()** - terminate a thread (POSIX)

**pthread\_equal()** - compare thread IDs (POSIX)

**pthread\_self()** - get the calling thread's ID (POSIX)

**pthread\_once()** - dynamic package initialization (POSIX)

**pthread\_key\_create()** - create a thread specific data key (POSIX)

**pthread\_setspecific()** - set thread specific data (POSIX)

**pthread\_getspecific()** - get thread specific data (POSIX)

**pthread\_key\_delete()** - delete a thread specific data key (POSIX)

**pthread\_cancel()** - cancel execution of a thread (POSIX)

**pthread\_setcancelstate()** - set cancellation state for calling thread (POSIX)

**pthread\_setcanceltype()** - set cancellation type for calling thread (POSIX)

**pthread\_testcancel()** - create a cancellation point in the calling thread (POSIX)

**pthread\_cleanup\_push()** - pushes a routine onto the cleanup stack (POSIX)

**pthread\_cleanup\_pop()** - pop a cleanup routine off the top of the stack (POSIX)

**DESCRIPTION**

This library provides an implementation of POSIX 1003.1c threads for VxWorks. This provides an increased level of compatibility between VxWorks applications and those written for other operating systems that support the POSIX threads model (often called *pthreads*).

VxWorks is a task based operating system, rather than one implementing the process model in the POSIX sense. As a result of this, there are a few restrictions in the implementation, but in general, since tasks are roughly equivalent to threads, the *pthreads*

support maps well onto VxWorks. The restrictions are explained in more detail in the following paragraphs.

- CONFIGURATION** To add POSIX threads support to a system, the component `INCLUDE_POSIX_PTHREADS` must be added.
- Threads support also requires the POSIX scheduler to be included (see `schedPxLib` for more detail).
- THREADS** A thread is essentially a VxWorks task, with some additional characteristics. The first is detachability, where the creator of a thread can optionally block until the thread exits. The second is cancelability, where one task or thread can cause a thread to exit, possibly calling cleanup handlers. The next is private data, where data private to a thread is created, accessed and deleted via keys. Each thread has a unique ID. A thread's ID is different than its VxWorks task ID.
- MUTEXES** Included with the POSIX threads facility is a mutual exclusion facility, or *mutex*. These are functionally similar to the VxWorks mutex semaphores (see `semMLib` for more detail), and in fact are implemented using a VxWorks mutex semaphore. The advantage they offer, like all of the POSIX libraries, is the ability to run software designed for POSIX platforms under VxWorks.
- There are two types of locking protocols available, `PTHREAD_PRIO_INHERIT` and `PTHREAD_PRIO_PROTECT`. `PTHREAD_PRIO_INHERIT` maps to a semaphore create with `SEM_PRIO_INHERIT` set (see `semMCreate` for more detail). A thread locking a mutex created with its protocol attribute set to `PTHREAD_PRIO_PROTECT` has its priority elevated to that of the *prioceiling* attribute of the mutex. When the mutex is unlocked, the priority of the calling thread is restored to its previous value.
- CONDITION VARIABLES**
- Condition variables are another synchronization mechanism that is included in the POSIX threads library. A condition variable allows threads to block until some condition is met. There are really only two basic operations that a condition variable can be involved in: waiting and signalling. Condition variables are always associated with a mutex.
- A thread can wait for a condition to become true by taking the mutex and then calling `pthread_cond_wait()`. That function will release the mutex and wait for the condition to be signalled by another thread. When the condition is signalled, the function will re-acquire the mutex and return to the caller.
- Condition variable support two types of signalling: single thread wake-up using `pthread_cond_signal()`, and multiple thread wake-up using `pthread_cond_broadcast()`. The latter of these will unblock all threads that were waiting on the specified condition variable.
- It should be noted that condition variable signals are not related to POSIX signals. In fact, they are implemented using VxWorks semaphores.

## RESOURCE COMPETITION

All tasks, and therefore all POSIX threads, compete for CPU time together. For that reason the contention scope thread attribute is always **PTHREAD\_SCOPE\_SYSTEM**.

## NO VXWORKS EQUIVALENT

Since there is no notion of a process (in the POSIX sense), there is no notion of sharing of locks (mutexes) and condition variables between processes. As a result, the POSIX symbol **\_POSIX\_THREAD\_PROCESS\_SHARED** is not defined in this implementation, and the routines **pthread\_condattr\_getpshared()**, **pthread\_condattr\_setpshared()**, **pthread\_mutexattr\_getpshared()** are not implemented.

Also, since there are no processes in VxWorks, **fork()**, **wait()**, and **pthread\_atfork()** are unimplemented.

VxWorks does not have password, user, or group databases, therefore there are no implementations of **getlogin()**, **getgrgid()**, **getpwnam()**, **getpwuid()**, **getlogin\_r()**, **getgrgid\_r()**, **getpwnam\_r()**, and **getpwuid\_r()**.

## SCHEDULING

The default scheduling policy for a created thread is inherited from the system setting at the time of creation.

Scheduling policies under VxWorks are global; they are not set per-thread, as the POSIX model describes. As a result, the *pthread* scheduling routines, as well as the POSIX scheduling routines native to VxWorks, do not allow you to change the scheduling policy. Under VxWorks you may set the scheduling policy in a thread, but if it does not match the system's scheduling policy, an error is returned.

The detailed explanation for why this error occurs is a bit convoluted: technically the scheduling policy is an attribute of a thread (in that there are **pthread\_attr\_getschedpolicy()** and **pthread\_attr\_setschedpolicy()** functions that define what the thread's scheduling policy will be once it is created, and not what any thread should do at the time they are called). A situation arises where the scheduling policy in force at the time of a thread's creation is not the same as set in its attributes. In this case **pthread\_create()** fails with an otherwise undocumented error **ENOTTY**.

The bottom line is that under VxWorks, if you wish to specify the scheduling policy of a thread, you must set the desired global scheduling policy to match. Threads must then adhere to that scheduling policy, or use the **PTHREAD\_INHERIT\_SCHED** mode to inherit the current mode and creator's priority.

## CREATION AND CANCELLATION

Each time a thread is created, the  *pthreads*  library allocates resources on behalf of it. Each time a VxWorks task (*i.e.*, one not created by the **pthread\_create()** function) uses a POSIX threads feature such as thread private data or pushes a cleanup handler, the  *pthreads*  library creates resources on behalf of that task as well.

Asynchronous thread cancellation is accomplished by way of a signal. A special signal, **SIGCANCEL**, has been set aside in this version of VxWorks for this purpose. Applications should take care not to block or handle **SIGCANCEL**.

**SUMMARY MATRIX**

<i>pthread</i> function	Implemented?	Note(s)
<i>pthread_attr_destroy</i>	Yes	
<i>pthread_attr_getdetachstate</i>	Yes	
<i>pthread_attr_getinheritsched</i>	Yes	
<i>pthread_attr_getschedparam</i>	Yes	
<i>pthread_attr_getschedpolicy</i>	Yes	
<i>pthread_attr_getscope</i>	Yes	
<i>pthread_attr_getstackaddr</i>	Yes	
<i>pthread_attr_getstacksize</i>	Yes	
<i>pthread_attr_init</i>	Yes	
<i>pthread_attr_setdetachstate</i>	Yes	
<i>pthread_attr_setinheritsched</i>	Yes	
<i>pthread_attr_setschedparam</i>	Yes	
<i>pthread_attr_setschedpolicy</i>	Yes	
<i>pthread_attr_setscope</i>	Yes	2
<i>pthread_attr_setstackaddr</i>	Yes	
<i>pthread_attr_setstacksize</i>	Yes	
<i>pthread_atfork</i>	No	1
<i>pthread_cancel</i>	Yes	5
<i>pthread_cleanup_pop</i>	Yes	
<i>pthread_cleanup_push</i>	Yes	
<i>pthread_condattr_destroy</i>	Yes	
<i>pthread_condattr_getpshared</i>	No	3
<i>pthread_condattr_init</i>	Yes	
<i>pthread_condattr_setpshared</i>	No	3
<i>pthread_cond_broadcast</i>	Yes	
<i>pthread_cond_destroy</i>	Yes	
<i>pthread_cond_init</i>	Yes	
<i>pthread_cond_signal</i>	Yes	
<i>pthread_cond_timedwait</i>	Yes	
<i>pthread_cond_wait</i>	Yes	
<i>pthread_create</i>	Yes	
<i>pthread_detach</i>	Yes	
<i>pthread_equal</i>	Yes	
<i>pthread_exit</i>	Yes	
<i>pthread_getschedparam</i>	Yes	4

P

<b>pthread function</b>	<b>Implemented?</b>	<b>Note(s)</b>
pthread_getspecific	Yes	
pthread_join	Yes	
pthread_key_create	Yes	
pthread_key_delete	Yes	
pthread_kill	Yes	
pthread_once	Yes	
pthread_self	Yes	
pthread_setcancelstate	Yes	
pthread_setcanceltype	Yes	
pthread_setschedparam	Yes	4
pthread_setspecific	Yes	
pthread_sigmask	Yes	
pthread_testcancel	Yes	
pthread_mutexattr_destroy	Yes	
pthread_mutexattr_getprioceiling	Yes	
pthread_mutexattr_getprotocol	Yes	
pthread_mutexattr_getpshared	No	3
pthread_mutexattr_init	Yes	
pthread_mutexattr_setprioceiling	Yes	
pthread_mutexattr_setprotocol	Yes	
pthread_mutexattr_setpshared	No	3
pthread_mutex_destroy	Yes	
pthread_mutex_getprioceiling	Yes	
pthread_mutex_init	Yes	
pthread_mutex_lock	Yes	
pthread_mutex_setprioceiling	Yes	
pthread_mutex_trylock	Yes	
pthread_mutex_unlock	Yes	
getlogin_r	No	6
getrgid_r	No	6
getpwnam_r	No	6
getpwuid_r	No	6

**NOTES**

- 1 The `pthread_atfork()` function is not implemented since `fork()` is not implemented in VxWorks.
- 2 The contention scope thread scheduling attribute is always `PTHREAD_SCOPE_SYSTEM`, since threads (*i.e.*, tasks) contend for resources with all other threads in the system.
- 3 The routines `pthread_condattr_getpshared()`, `pthread_attr_setpshared()`, `pthread_mutexattr_getpshared()` and `pthread_mutexattr_setpshared()` are not

supported, since these interfaces describe how condition variables and mutexes relate to a process, and VxWorks does not implement a process model.

- 4 The default scheduling policy is inherited from the current system setting. The POSIX model of per-thread scheduling policies is not supported, since a basic tenet of the design of VxWorks is a system-wide scheduling policy.
- 5 Thread cancellation is supported in appropriate *pthread* routines and those routines already supported by VxWorks. However, the complete list of cancellation points specified by POSIX is not supported because routines such as **msync()**, **fcntl()**, **tcdrain()**, and **wait()** are not implemented by VxWorks.
- 6 The routines **getlogin\_r()**, **getgrgid\_r()**, **getpwnam\_r()**, and **getpwuid\_r()** are not implemented.

**INCLUDE FILES**     **pthread.h**

**SEE ALSO**            **taskLib**, **semMLib**, **semPxLib**, *VxWorks Programmer's Guide: Multitasking*

---

## ptyDrv

**NAME**                **ptyDrv** – pseudo-terminal driver

**ROUTINES**          **ptyDrv()** - initialize the pseudo-terminal driver  
**ptyDevCreate()** - create a pseudo terminal  
**ptyDevRemove()** - destroy a pseudo terminal  
**ptyShow()** - show the state of the Pty Buffers

**DESCRIPTION**      The pseudo-terminal driver provides a *tty*-like interface between a master and slave process, typically in network applications. The master process simulates the “hardware” side of the driver (*e.g.*, a USART serial chip), while the slave process is the application program that normally talks to the driver.

**USER-CALLABLE ROUTINES**

Most of the routines in this driver are accessible only through the I/O system. However, the following routines must be called directly: **ptyDrv()** to initialize the driver, **ptyDevCreate()** to create devices, and **ptyDevRemove()** to remove an existing device.

**INITIALIZING THE DRIVER**

Before using the driver, it must be initialized by calling **ptyDrv()**. This routine must be called before any reads, writes, or calls to **ptyDevCreate()**.

#### CREATING PSEUDO-TERMINAL DEVICES

Before a pseudo-terminal can be used, it must be created by calling **ptyDevCreate()**:

```
STATUS ptyDevCreate
(
    char *name,      /* name of pseudo terminal */
    int  rdBufSize, /* size of terminal read buffer */
    int  wrtBufSize /* size of write buffer */
)
```

For instance, to create the device pair **/pty/0.M** and **/pty/0.S**, with read and write buffer sizes of 512 bytes, the proper call would be:

```
ptyDevCreate ("/pty/0.", 512, 512);
```

When **ptyDevCreate()** is called, two devices are created, a master and slave. One is called *nameM* and the other *nameS*. They can then be opened by the master and slave processes. Data written to the master device can then be read on the slave device, and vice versa. Calls to **ioctl()** may be made to either device, but they should only apply to the slave side, since the master and slave are the same device.

The **ptyDevRemove()** routine will delete an existing pseudo-terminal device and reclaim the associated memory. Any file descriptors associated with the device will be closed.

**IOCTL FUNCTIONS** Pseudo-terminal drivers respond to the same **ioctl()** functions used by *tty* devices. These functions are defined in **ioLib.h** and documented in the manual entry for **tyLib**.

**INCLUDE FILES** **ioLib.h**, **ptyDrv.h**

**SEE ALSO** **tyLib**, *VxWorks Programmer's Guide: I/O System*

---

## ramDiskCbio

<b>NAME</b>	<b>ramDiskCbio</b> – RAM Disk Cached Block Driver
<b>ROUTINES</b>	<b>ramDiskDevCreate()</b> - Initialize a RAM Disk device
<b>DESCRIPTION</b>	<p>This module implements a RAM-disk driver with a CBIO interface which can be directly utilized by <b>dosFsLib</b> without the use of the Disk Cache module <b>dcacheCbio</b>. This results in an ultra-compact RAM footprint. This module is implemented using the CBIO API (see <b>cbioLib()</b>)</p> <p>This module is delivered in source as a functional example of a basic CBIO module.</p> <hr/> <p><b>WARNING:</b> This module may be used for SRAM or other non-volatile RAM cards to store a file system, but that configuration will be susceptible to data corruption in events of system failure which are not normally observed with magnetic disks, <i>i.e.</i>, using this driver with an SRAM card can not guard against interruptions in midst of updating a particular sector, resulting in that sector become internally inconsistent.</p> <hr/>
<b>SEE ALSO</b>	<b>dosFsLib</b> , <b>cbioLib</b>

---

## ramDrv

<b>NAME</b>	<b>ramDrv</b> – RAM disk driver
<b>ROUTINES</b>	<b>ramDrv()</b> - prepare a RAM disk driver for use (optional) <b>ramDevCreate()</b> - create a RAM disk device
<b>DESCRIPTION</b>	<p>This driver emulates a disk driver, but actually keeps all data in memory. The memory location and size are specified when the “disk” is created. The RAM disk feature is useful when data must be preserved between boots of VxWorks or when sharing data between CPUs.</p>
<b>USER-CALLABLE ROUTINES</b>	<p>Most of the routines in this driver are accessible only through the I/O system. Two routines, however, can be called directly by the user. The first, <b>ramDrv()</b>, provides no real function except to parallel the initialization function found in true disk device drivers. A call to <b>ramDrv()</b> is not required to use the RAM disk driver. However, the second routine, <b>ramDevCreate()</b>, must be called directly to create RAM disk devices.</p>

Once the device has been created, it must be associated with a name and file system (dosFs, rt11Fs, or rawFs). This is accomplished by passing the value returned by **ramDevCreate()**, a pointer to a block device structure, to the file system's device initialization routine or make-file-system routine. See the manual entry **ramDevCreate()** for a more detailed discussion.

**IOCTL FUNCTIONS** The RAM driver is called in response to **ioctl()** codes in the same manner as a normal disk driver. When the file system is unable to handle a specific **ioctl()** request, it is passed to the **ramDrv** driver. Although there is no physical device to be controlled, **ramDrv** does handle a **FIODISKFORMAT** request, which always returns OK. All other **ioctl()** requests return an error and set the task's **errno** to **S\_ioLib\_UNKNOWN\_REQUEST**.

**INCLUDE FILE** **ramDrv.h**

**SEE ALSO** **dosFsDevInit()**, **dosFsMkfs()**, **rt11FsDevInit()**, **rt11FsMkfs()**, **rawFsDevInit()**, *VxWorks Programmer's Guide: I/O System, Local File Systems*

---

## rawFsLib

**NAME** **rawFsLib** – raw block device file system library

**ROUTINES** **rawFsDevInit()** - associate a block device with raw volume functions  
**rawFsInit()** - prepare to use the raw volume library  
**rawFsModeChange()** - modify the mode of a raw device volume  
**rawFsReadyChange()** - notify **rawFsLib** of a change in ready status  
**rawFsVolUnmount()** - disable a raw device volume

**DESCRIPTION** This library provides basic services for disk devices that do not use a standard file or directory structure. The disk volume is treated much like a large file. Portions of it may be read, written, or the current position within the disk may be changed. However, there is no high-level organization of the disk into files or directories.

### USING THIS LIBRARY

The various routines provided by the VxWorks raw "file system" (rawFs) may be separated into three broad groups: general initialization, device initialization, and file system operation.

The **rawFsInit()** routine is the principal initialization function; it need only be called once, regardless of how many rawFs devices will be used.

A separate rawFs routine is used for device initialization. For each rawFs device, **rawFsDevInit()** must be called to install the device.

Several routines are provided to inform the file system of changes in the system environment. The **rawFsModeChange()** routine may be used to modify the readability or writability of a particular device. The **rawFsReadyChange()** routine is used to inform the file system that a disk may have been swapped and that the next disk operation should first remount the disk. The **rawFsVolUnmount()** routine informs the file system that a particular device should be synchronized and unmounted, generally in preparation for a disk change.

**INITIALIZATION** Before any other routines in **rawFsLib** can be used, **rawFsInit()** must be called to initialize the library. This call specifies the maximum number of raw device file descriptors that can be open simultaneously and allocates memory for that many raw file descriptors. Any attempt to open more raw device file descriptors than the specified maximum will result in errors from **open()** or **creat()**.

During the **rawFsInit()** call, the raw device library is installed as a driver in the I/O system driver table. The driver number associated with it is then placed in a global variable, **rawFsDrvNum**.

This initialization is enabled when the configuration macro **INCLUDE\_RAWFS** is defined; **rawFsInit()** is then called from the root task, **usrRoot()**, in **usrConfig.c**.

#### DEFINING A RAW DEVICE

To use this library for a particular device, the device structure used by the device driver must contain, as the very first item, a CPIO device description structure (**CBIO\_DEV**) or block device description structure (**BLK\_DEV**). This must be initialized before calling **rawFsDevInit()**.

The **rawFsDevInit()** routine is used to associate a device with the **rawFsLib** functions. The *pVolName* parameter expected by **rawFsDevInit()** is a pointer to a name string, to be used to identify the device. This will serve as the pathname for I/O operations which operate on the device. This name will appear in the I/O system device table, which may be displayed using **iosDevShow()**.

The syntax of the **rawFsDevInit()** routine is as follows:

```
rawFsDevInit
(
    char      *pVolName, /* name to be used for volume - iosDevAdd */
    BLK_DEV  *pDevice   /* pointer to BLK_DEV device or a CBIO_DEV_ID */
)
```

Unlike the VxWorks DOS file system, raw volumes do not require an **FIODISKINIT ioctl()** function to initialize volume structures. (Such an **ioctl()** call can be made for a raw volume, but it has no effect.) As a result, there is no “make file system” routine for raw volumes (for comparison, see the manual entry for **rawFsMkfs()**).

When **rawFsLib** receives a request from the I/O system, after **rawFsDevInit()** has been called, it calls the appropriate device driver routines to access the device.

#### **MULTIPLE LOGICAL DEVICES**

The block number passed to the block read and write routines is an absolute number, starting from block 0 at the beginning of the device. If desired, the driver may add an offset from the beginning of the physical device before the start of the logical device. This would normally be done by keeping an offset parameter in the driver's device-specific structure, and adding the proper number of blocks to the block number passed to the read and write routines. See the **ramDrv** manual entry for an example.

#### **UNMOUNTING VOLUMES (CHANGING DISKS)**

A disk should be unmounted before it is removed. When unmounted, any modified data that has not been written to the disk will be written out. A disk may be unmounted by either calling **rawFsVolUnmount()** directly or calling **ioctl()** with a **FIODISKCHANGE** function code.

There may be open file descriptors to a raw device volume when it is unmounted. If this is the case, those file descriptors will be marked as obsolete. Any attempts to use them for further I/O operations will return an **S\_rawFsLib\_FD\_OBSOLETE** error. To free such file descriptors, use the **close()** call, as usual. This will successfully free the descriptor, but will still return **S\_rawFsLib\_FD\_OBSOLETE**.

#### **SYNCHRONIZING VOLUMES**

A disk should be "synchronized" before it is unmounted. To synchronize a disk means to write out all buffered data (the write buffers associated with open file descriptors), so that the disk is updated. It may or may not be necessary to explicitly synchronize a disk, depending on how (or if) the driver issues the **rawFsVolUnmount()** call.

When **rawFsVolUnmount()** is called, an attempt will be made to synchronize the device before unmounting. However, if the **rawFsVolUnmount()** call is made by a driver in response to a disk being removed, it is obviously too late to synchronize. Therefore, a separate **ioctl()** call specifying the **FIOSYNC** function should be made before the disk is removed. (This could be done in response to an operator command.)

If the disk will still be present and writable when **rawFsVolUnmount()** is called, it is not necessary to first synchronize the disk. In all other circumstances, failure to synchronize the volume before unmounting may result in lost data.

**IOCTL FUNCTIONS** The VxWorks raw block device file system supports the following **ioctl()** functions. The functions listed are defined in the header **ioLib.h**.

##### **FIODISKFORMAT**

No file system is initialized on the disk by this request. This **ioctl** is passed directly down to the driver-provided function:

```
fd = open ("DEV1:", O_WRONLY);
status = ioctl (fd, FIODISKFORMAT, 0);
```

##### **FIODISKINIT**

Initializes a raw file system on the disk volume. Since there are no file system

structures, this functions performs no action. It is provided only for compatibility with other VxWorks file systems.

**FIODISKCHANGE**

Announces a media change. It performs the same function as **rawFsReadyChange()**. This function may be called from interrupt level:

```
status = ioctl (fd, FIODISKCHANGE, 0);
```

**FIOUNMOUNT**

Unmounts a disk volume. It performs the same function as **rawFsVolUnmount()**. This function must not be called from interrupt level:

```
status = ioctl (fd, FIOUNMOUNT, 0);
```

**FIOGETNAME**

Gets the file name of the file descriptor and copies it to the buffer *nameBuf*:

```
status = ioctl (fd, FIOGETNAME, &nameBuf);
```

**FIOSEEK**

Sets the current byte offset on the disk to the position specified by *newOffset*:

```
status = ioctl (fd, FIOSEEK, newOffset);
```

**FIOWHERE**

Returns the current byte position from the start of the device for the specified file descriptor. This is the byte offset of the next byte to be read or written. It takes no additional argument:

```
position = ioctl (fd, FIOWHERE, 0);
```

**FIOFLUSH**

Writes all modified file descriptor buffers to the physical device.

```
status = ioctl (fd, FIOFLUSH, 0);
```

**FIOSYNC**

Performs the same function as **FIOFLUSH**.

**FIONREAD**

Copies to *unreadCount* the number of bytes from the current file position to the end of the device:

```
status = ioctl (fd, FIONREAD, &unreadCount);
```

**INCLUDE FILES**     **rawFsLib.h**

**SEE ALSO**         **ioLib, iosLib, rawFsLib, ramDrv, VxWorks Programmer's Guide: I/O System, Local File Systems**

## rBuffLib

<b>NAME</b>	<b>rBuffLib</b> – dynamic ring buffer (rBuff) library
<b>ROUTINES</b>	<b>wvRBuffMgrPrioritySet()</b> - set the priority of the WindView rBuff manager (WindView)
<b>DESCRIPTION</b>	This library contains a routine for changing the default priority of the rBuff manager task.
<b>SEE ALSO</b>	<b>memLib</b> , <b>rngLib</b> , <i>VxWorks Programmer's Guide: Basic OS</i>

---

## rdiscLib

<b>NAME</b>	<b>rdiscLib</b> – ICMP router discovery server library
<b>ROUTINES</b>	<b>rdiscLibInit()</b> - Initialize router discovery <b>rdiscInit()</b> - initialize the ICMP router discovery function <b>sendAdvert()</b> - send an advertisement to one location <b>sendAdvertAll()</b> - send an advertisement to all active locations <b>rdiscTimerEvent()</b> - called after watchdog timeout <b>rdisc()</b> - implement the ICMP router discovery function <b>rdCtl()</b> - implement the ICMP router discovery control function <b>rdiscIfReset()</b> - check for new or removed interfaces for router discovery
<b>DESCRIPTION</b>	<p><b>rdiscLib</b> contains code to implement ICMP Router Discovery. This feature allows routers to advertise an address to the hosts on each of the routers interfaces. This address is placed by the host into its route table as a default router. A host may also solicit the address by multicasting the request to the <b>ALL_ROUTERS</b> address (224.0.0.2), to which a router would respond with a unicast version of the advertisement.</p> <p>There are three routines in this implementation of router discovery: <b>rdiscInit()</b>, <b>rdisc()</b> and <b>rdCtl()</b>. <b>rdiscInit()</b> is the initialization routine, <b>rdisc()</b> handles the periodic transmission of advertisements and processing of solicitations, and <b>rdCtl()</b> sets/gets user parameters.</p>

---

## rebootLib

<b>NAME</b>	<b>rebootLib</b> – reboot support library
<b>ROUTINES</b>	<b>reboot()</b> - reset network devices and transfer control to boot ROMs <b>rebootHookAdd()</b> - add a routine to be called at reboot
<b>DESCRIPTION</b>	This library provides reboot support. To restart VxWorks, the routine <b>reboot()</b> can be called at any time by typing CTRL-X from the shell. Shutdown routines can be added with <b>rebootHookAdd()</b> . These are typically used to reset or synchronize hardware. For example, <b>netLib</b> adds a reboot hook to cause all network interfaces to be reset. Once the reboot hooks have been run, <b>sysToMonitor()</b> is called to transfer control to the boot ROMs. For more information, see the manual entry for <b>bootInit</b> .
<b>DEFICIENCIES</b>	The order in which hooks are added is the order in which they are run. As a result, <b>netLib</b> will kill the network, and no user-added hook routines will be able to use the network. There is no <b>rebootHookDelete()</b> routine.
<b>INCLUDE FILES</b>	<b>rebootLib.h</b>
<b>SEE ALSO</b>	<b>sysLib</b> , <b>bootConfig</b> , <b>bootInit</b>

---

## remLib

<b>NAME</b>	<b>remLib</b> – remote command library
<b>ROUTINES</b>	<b>rcmd()</b> - execute a shell command on a remote machine <b>rresvport()</b> - open a socket with a privileged port bound to it <b>remCurIdGet()</b> - get the current user name and password <b>remCurIdSet()</b> - set the remote user name and password <b>iam()</b> - set the remote user name and password <b>whoami()</b> - display the current remote identity <b>bindresvport()</b> - bind a socket to a privileged IP port
<b>DESCRIPTION</b>	This library provides routines that support remote command functions. The <b>rcmd()</b> and <b>rresvport()</b> routines use protocols implemented in BSD 4.3; they support remote command execution, and the opening of a socket with a bound privileged port, respectively. For more information, see <i>Unix Network Programming</i> by W. Richard Stevens. This library also includes routines that authorize network file access via <b>netDrv</b> .

To include **remLib** in a VxWorks image, include the **NETWRS\_REMLIB** configuration component. This component contains one parameter, **RSH\_STDERR\_SETUP\_TIMEOUT**. Use this parameter to specify how long an **rcmd()** call should wait for a return from its internal call to **select()**. Valid values for **RSH\_STDERR\_SETUP\_TIMEOUT** are 0 (**NO\_WAIT**), -1 (**WAIT\_FOREVER**), or a positive integer from 1 to 2147483647 inclusive. This positive integer specifies the wait in seconds. The default value for **RSH\_STDERR\_SETUP\_TIMEOUT** is -1 (**WAIT\_FOREVER**).

**INCLUDE FILES**     **remLib.h**

**SEE ALSO**         **inetLib**

---

## remShellLib

**NAME**             **remShellLib** – remote access to target shell

**ROUTINES**        No Callable Routines

**DESCRIPTION**    This library contains the support routines for remote access to the VxWorks target shell for clients using the **telnet** or **rlogin** protocols. It supplies file descriptors to connection **telnet** or **rlogin** sessions to the shell's command interpreter.

**INCLUDE FILES**   **remShellLib.h, shellLib.h**

---

## resolvLib

**NAME**             **resolvLib** – DNS resolver library

**ROUTINES**        **resolvInit()** - initialize the resolver library  
**resolvGetHostByName()** - query the DNS server for the IP address of a host  
**resolvGetHostByAddr()** - query the DNS server for the host name of an IP address  
**resolvParamsSet()** - set the parameters which control the resolver library  
**resolvParamsGet()** - get the parameters which control the resolver library  
**resolvDNExpand()** - expand a DNS compressed name from a DNS packet  
**resolvDNComp()** - compress a DNS name in a DNS packet  
**resolvQuery()** - construct a query, send it, wait for a response  
**resolvMkQuery()** - create all types of DNS queries  
**resolvSend()** - send a pre-formatted query and return the answer

**DESCRIPTION**

This library provides the client-side services for DNS (Domain Name Service) queries. DNS queries come from applications that require translation of IP addresses to host names and back. If you include this library in VxWorks, it extends the services of the host library. The interface to this library is described in **hostLib**. The **hostLib** interface uses resolver services to get IP and host names. In addition, the resolver can query multiple DNS servers, if necessary, to add redundancy for queries.

There are two interfaces available for the resolver library. One is a high-level interface suitable for most applications. The other is also a low-level interface for more specialized applications, such as mail protocols.

**USING THIS LIBRARY**

By default, a VxWorks build does not include the resolver code. In addition, VxWorks is delivered with the resolver library disabled. To include the resolver library in the VxWorks image, edit **config/all/configAll.h** and include the definition:

```
#define INCLUDE_DNS_RESOLVER
```

To enable the resolver services, you need to redefine only one DNS server IP address, changing it from a place-holder value to an actual value. Additional DNS server IP addresses can be configured using **resolvParamsSet()**. To do the initial configuration, edit **configAll.h**, and enter the correct IP address for your domain server in the definition:

```
#define RESOLVER_DOMAIN_SERVER "90.0.0.3"
```

If you do not provide a valid IP address, resolver initialization fails. You also need to configure the domain to which your resolver belongs. To do this, edit **configAll.h** and enter the correct domain name for your organization in the definition:

```
#define RESOLVER_DOMAIN "wrs.com"
```

The last and most important step is to make sure that you have a route to the configured DNS server. If your VxWorks image includes a routing protocol, such as RIP or OSPF, the routes are created for you automatically. Otherwise, you must use **routeAdd()** or **mRouteAdd()** to add the routes to the routing table.

The resolver library comes with a debug option. To turn on debugging, edit **configAll.h** to include the define:

```
#define INCLUDE_DNS_DEBUG
```

This include makes VxWorks print a log of the resolver queries to the console. This feature assumes a single task. Thus, if you are running multiple tasks, your output to the console is a garble of messages from all the tasks.

The resolver library uses UDP to send queries to the DNS server and expects the DNS server to handle recursion. You can change the resolver parameters at any time after the library has been initialized with **resolvInit()**. However, it is strongly recommended that you change parameters only shortly after initialization, or when there are no other tasks accessing the resolver library.

Your procedure for changing any of the resolver parameter should start with a call to **resolvParamsGet()** to retrieve the active parameters. Then you can change the query order (defaults to query DNS server only), the domain name, or add DNS server IP addresses. After the parameters are changed, call **resolvParamsSet()**. For the values you can use when accessing resolver library services, see the header files **resolvLib.h**, **resolv/resolv.h**, and **resolv/nameser.h**.

**INCLUDE FILES**     **resolvLib.h**

**SEE ALSO**         **hostLib**

---

## ripLib

**NAME**             **ripLib** – Routing Information Protocol (RIP) v1 and v2 library

**ROUTINES**

- ripLibInit()** - initialize the RIP routing library
- ripAddrsXtract()** - extract socket address pointers from the route message
- ripRouteShow()** - display the internal routing table maintained by RIP
- ripIfShow()** - display the internal interface table maintained by RIP
- ripAuthHookAdd()** - add an authentication hook to a RIP interface
- ripAuthHookDelete()** - remove an authentication hook from a RIP interface
- ripAuthHook()** - sample authentication hook
- ripLeakHookAdd()** - add a hook to bypass the RIP and kernel routing tables
- ripLeakHookDelete()** - remove a table bypass hook from a RIP interface
- ripSendHookAdd()** - add an update filter to a RIP interface
- ripSendHookDelete()** - remove an update filter from a RIP interface
- ripRouteHookAdd()** - add a hook to install static and non-RIP routes into RIP
- ripRouteHookDelete()** - remove the route hook
- ripIfSearch()** - add new interfaces to the internal list
- ripIfReset()** - alter the RIP configuration after an interface changes
- ripFilterEnable()** - activate strict border gateway filtering
- ripFilterDisable()** - prevent strict border gateway filtering
- ripShutdown()** - terminate all RIP processing
- ripDebugLevelSet()** - specify amount of debugging output
- ripAuthKeyShow()** - show current authentication configuration
- ripAuthKeyAdd()** - add a new RIP authentication key
- ripAuthKeyDelete()** - delete an existing RIP authentication key
- ripAuthKeyFind()** - find a RIP authentication key
- ripAuthKeyFindFirst()** - find a RIP authentication key
- ripAuthKeyInMD5()** - authenticate an incoming RIP-2 message using MD5
- ripAuthKeyOut1MD5()** - start MD5 authentication of an outgoing RIP-2 message
- ripAuthKeyOut2MD5()** - authenticate an outgoing RIP-2 message using MD5
- ripIfExcludeListAdd()** - Add an interface to the RIP exclusion list

**ripIfExcludeListDelete()** - Delete an interface from RIP exclusion list  
**ripIfExcludeListShow()** - Show the RIP interface exclusion list

**DESCRIPTION** This library implements versions 1 and 2 of the Routing Information Protocol (RIP). The protocol is intended to operate as an interior gateway protocol within a relatively small network with a longest path of 15 hops.

#### HIGH-LEVEL INTERFACE

The **ripLibInit()** routine links this library into the VxWorks image and begins a RIP session. This happens automatically if **INCLUDE\_RIP** is defined at the time the image is built. Once started, RIP will maintain the network routing table until deactivated by a call to the **ripShutdown()** routine, which will remove all route entries and disable the RIP library routines. All RIP requests and responses are handled as defined in the RFC specifications. RFC 1058 defines the basic protocol operation and RFC 1723 details the extensions that constitute version 2.

When acting as a supplier, outgoing route updates are filtered using simple split horizon. Split horizon with poisoned reverse is not currently available. Additional route entries may be excluded from the periodic update with the **ripSendHookAdd()** routine.

If a RIP session is terminated, the networking subsystem may not function correctly until RIP is restarted with a new call to **ripLibInit()** unless routing information is provided by some other method.

#### CONFIGURATION INTERFACE

By default, a RIP session only uses the network interfaces created before it started. The **ripIfSearch()** routine allows RIP to recognize any interfaces added to the system after that point. If the address or netmask of an existing interface is changed during a RIP session, the **ripIfReset()** routine must be used to update the RIP configuration appropriately. The current RIP implementation also automatically performs the border gateway filtering required by the RFC specification. Those restrictions provide correct operation in a mixed environment of RIP-1 and RIP-2 routers. The **ripFilterDisable()** routine will remove those limitations, and can produce more efficient routing for some topologies. However, you must not use that routine if any version 1 routers are present. The **ripFilterEnable()** routine will restore the default behavior.

#### AUTHENTICATION INTERFACE

By default, authentication is disabled, but may be activated by an SNMP agent on an interface-specific basis. While authentication is disabled, any RIP-2 messages containing authentication entries are discarded. When enabled, all RIP-2 messages without authentication entries are automatically rejected. To fully support authentication, an authentication routine should be specified with the **ripAuthHookAdd()** routine. The specified function will be called to screen every RIP-1 message and all unverified RIP-2 messages containing authentication entries. It may be removed with the **ripAuthHookDelete()** routine. All RIP-1 and unverified RIP-2 messages will be discarded while authentication is enabled unless a hook is present.

#### OPTIONAL INTERFACE

The **ripLeakHookAdd()** routine allows the use of an alternative routing protocol that uses RIP as a transport mechanism. The specified function can prevent the RIP session from creating any table entries from the received messages. The **ripLeakHookDelete()** routine will restore the default operation.

#### DEBUGGING INTERFACE

As required by the RFC specification, the obsolete **traceon** and **traceoff** messages are not supported by this implementation. The **ripRouteShow()** routine will display the contents of the internal RIP routing table. Routines such as **mRouteShow()** to display the corresponding kernel routing table will also be available if **INCLUDE\_NET\_SHOW** is defined when the image is built. If additional information is required, the **ripDebugLevelSet()** routine will enable predefined debugging messages that will be sent to the standard output.

**INCLUDE FILES**     **ripLib.h**

**SEE ALSO**            RFC 1058, RFC 1723

---

## rlogLib

**NAME**                **rlogLib** – remote login library

**ROUTINES**          **rlogInit()** - initialize the remote login facility  
**rlogind()** - the VxWorks remote login daemon  
**rlogin()** - log in to a remote host

**DESCRIPTION**      This library provides a remote login facility for VxWorks based on the UNIX **rlogin** protocol (as implemented in UNIX BSD 4.3). On a VxWorks terminal, this command gives users the ability to log in to remote systems on the network.

Reciprocally, the remote login daemon, **rlogind()**, allows remote users to log in to VxWorks. The daemon is started by calling **rlogInit()**, which is called automatically when **INCLUDE\_RLOGIN** is defined. The remote login daemon accepts remote login requests from another VxWorks or UNIX system, and causes the shell's input and output to be redirected to the remote user.

Internally, **rlogind()** provides a *tty*-like interface to the remote user through the use of the VxWorks pseudo-terminal driver **ptyDrv**.

**INCLUDE FILES**     **rlogLib.h**

**SEE ALSO**            **ptyDrv**, **telnetLib**, UNIX BSD 4.3 manual entries for **rlogin**, **rlogind**, and **pty**

---

## rngLib

**NAME**            **rngLib** – ring buffer subroutine library

**ROUTINES**        **rngCreate()** - create an empty ring buffer  
**rngDelete()** - delete a ring buffer  
**rngFlush()** - make a ring buffer empty  
**rngBufGet()** - get characters from a ring buffer  
**rngBufPut()** - put bytes into a ring buffer  
**rngIsEmpty()** - test if a ring buffer is empty  
**rngIsFull()** - test if a ring buffer is full (no more room)  
**rngFreeBytes()** - determine the number of free bytes in a ring buffer  
**rngNBytes()** - determine the number of bytes in a ring buffer  
**rngPutAhead()** - put a byte ahead in a ring buffer without moving ring pointers  
**rngMoveAhead()** - advance a ring pointer by *n* bytes

**DESCRIPTION**    This library provides routines for creating and using ring buffers, which are first-in-first-out circular buffers. The routines simply manipulate the ring buffer data structure; no kernel functions are invoked. In particular, ring buffers by themselves provide no task synchronization or mutual exclusion.

However, the ring buffer pointers are manipulated in such a way that a reader task (invoking **rngBufGet()**) and a writer task (invoking **rngBufPut()**) can access a ring simultaneously without requiring mutual exclusion. This is because readers only affect a *read* pointer and writers only affect a *write* pointer in a ring buffer data structure. However, access by multiple readers or writers *must* be interlocked through a mutual exclusion mechanism (*i.e.*, a mutual-exclusion semaphore guarding a ring buffer).

This library also supplies two macros, **RNG\_ELEM\_PUT** and **RNG\_ELEM\_GET**, for putting and getting single bytes from a ring buffer. They are defined in **rngLib.h**.

```
int RNG_ELEM_GET (ringId, pch, fromP)
int RNG_ELEM_PUT (ringId, ch, toP)
```

Both macros require a temporary variable *fromP* or *toP*, which should be declared as **register int** for maximum efficiency. **RNG\_ELEM\_GET** returns 1 if there was a character available in the buffer; it returns 0 otherwise. **RNG\_ELEM\_PUT** returns 1 if there was room in the buffer; it returns 0 otherwise. These are somewhat faster than **rngBufPut()** and **rngBufGet()**, which can put and get multi-byte buffers.

**INCLUDE FILES**   **rngLib.h**

## routeEntryLib

<b>NAME</b>	<b>routeEntryLib</b> – route interface library for multiple matching entries
<b>ROUTINES</b>	<b>routeModify()</b> - change an entry in the routing table <b>routeEntryAdd()</b> - insert a route in the routing table <b>routeEntryDel()</b> - remove a route from the routing table <b>routeTableWalk()</b> - traverse the IP routing table <b>routeEntryLookup()</b> - find a matching route for a destination

---

## routeLib

<b>NAME</b>	<b>routeLib</b> – network route manipulation library
<b>ROUTINES</b>	<b>routeAdd()</b> - add a route <b>routeNetAdd()</b> - add a route to a destination that is a network <b>routeDelete()</b> - delete a route <b>mRouteAdd()</b> - add multiple routes to the same destination <b>mRouteEntryAdd()</b> - add a protocol-specific route to the routing table <b>mRouteEntryDelete()</b> - delete route from the routing table <b>mRouteDelete()</b> - delete a route from the routing table
<b>DESCRIPTION</b>	This library contains the routines for inspecting the routing table, as well as routines for adding and deleting routes from that table. If you do not configure VxWorks to include a routing protocol, such as RIP or OSPF, you can use these routines to maintain the routing tables manually.  To use this feature, include the following component: <b>INCLUDE_NETWRS_ROUTELIB</b>
<b>INCLUDE FILES</b>	<b>routeLib.h</b>
<b>SEE ALSO</b>	<b>hostLib</b>

---

## routeMessageLib

<b>NAME</b>	<b>routeMessageLib</b> – message routines for the routing interface library
<b>ROUTINES</b>	<b>routeStorageUnbind()</b> - remove a registered handler from the routing system

---

## rpcLib

<b>NAME</b>	<b>rpcLib</b> – Remote Procedure Call (RPC) support library
<b>ROUTINES</b>	<b>rpcInit()</b> - initialize the RPC package <b>rpcTaskInit()</b> - initialize a task's access to the RPC package
<b>DESCRIPTION</b>	<p>This library supports Sun Microsystems' Remote Procedure Call (RPC) facility. RPC provides facilities for implementing distributed client/server-based architectures. The underlying communication mechanism can be completely hidden, permitting applications to be written without any reference to network sockets. The package is structured such that lower-level routines can optionally be accessed, allowing greater control of the communication protocols.</p> <p>For more information and a tutorial on RPC, see Sun Microsystems' <i>Remote Procedure Call Programming Guide</i>. For an example of RPC usage, see <b>/target/unsupported/demo/sprites</b>.</p> <p>The RPC facility is enabled when <b>INCLUDE_RPC</b> is defined.</p> <p>VxWorks supports Network File System (NFS), which is built on top of RPC. If NFS is configured into the VxWorks system, RPC is automatically included as well.</p>
<b>IMPLEMENTATION</b>	<p>A task must call <b>rpcTaskInit()</b> before making any calls to other routines in the RPC library. This routine creates task-specific data structures required by RPC. These task-specific data structures are automatically deleted when the task exits.</p> <p>Because each task has its own RPC context, RPC-related objects (such as SVCXPRTs and CLIENTs) cannot be shared among tasks; objects created by one task cannot be passed to another for use. Such additional objects must be explicitly deleted (for example, using task deletion hooks).</p>
<b>INCLUDE FILES</b>	<b>rpc.h</b>
<b>SEE ALSO</b>	<b>nfsLib</b> , <b>nfsDrv</b> , Sun Microsystems' <i>Remote Procedure Call Programming Guide</i>

---

## rt11FsLib

<b>NAME</b>	<b>rt11FsLib</b> – RT-11 media-compatible file system library
<b>ROUTINES</b>	<b>rt11FsDevInit()</b> - initialize the rt11Fs device descriptor <b>rt11FsInit()</b> - prepare to use the rt11Fs library <b>rt11FsMkfs()</b> - initialize a device and create an rt11Fs file system

**rt11FsDataSet()** - set the rt11Fs file system date  
**rt11FsReadyChange()** - notify rt11Fs of a change in ready status  
**rt11FsModeChange()** - modify the mode of an rt11Fs volume

**DESCRIPTION** This library provides services for file-oriented device drivers which use the RT-11 file standard. This module takes care of all the necessary buffering, directory maintenance, and RT-11-specific details.

#### USING THIS LIBRARY

The various routines provided by the VxWorks RT-11 file system (rt11Fs) may be separated into three broad groups: general initialization, device initialization, and file system operation.

The **rt11FsInit()** routine is the principal initialization function; it need only be called once, regardless of how many rt11Fs devices will be used.

Other rt11Fs routines are used for device initialization. For each rt11Fs device, either **rt11FsDevInit()** or **rt11FsMkfs()** must be called to install the device and define its configuration.

Several functions are provided to inform the file system of changes in the system environment. The **rt11FsDataSet()** routine is used to set the date. The **rt11FsModeChange()** routine is used to modify the readability or writability of a particular device. The **rt11FsReadyChange()** routine is used to inform the file system that a disk may have been swapped, and that the next disk operation should first remount the disk.

#### INITIALIZING RT11FSLIB

Before any other routines in **rt11FsLib** can be used, **rt11FsInit()** must be called to initialize this library. This call specifies the maximum number of rt11Fs files that can be open simultaneously and allocates memory for that many rt11Fs file descriptors. Attempts to open more files than the specified maximum will result in errors from **open()** or **creat()**.

This initialization is enabled when the configuration macro **INCLUDE\_RT11FS** is defined.

#### DEFINING AN RT-11 DEVICE

To use this library for a particular device, the device structure must contain, as the very first item, a **BLK\_DEV** structure. This must be initialized before calling **rt11FsDevInit()**. In the **BLK\_DEV** structure, the driver includes the addresses of five routines which it must supply: one that reads one or more sectors, one that writes one or more sectors, one that performs I/O control on the device (using **ioctl()**), one that checks the status of the device, and one that resets the device. This structure also specifies various physical aspects of the device (e.g., number of sectors, sectors per track, whether the media is removable). For more information about defining block devices, see the *VxWorks Programmer's Guide: I/O System*.

The device is associated with the rt11Fs file system by the **rt11FsDevInit()** call. The arguments to **rt11FsDevInit()** include the name to be used for the rt11Fs volume, a

pointer to the **BLK\_DEV** structure, whether the device uses RT-11 standard skew and interleave, and the maximum number of files that can be contained in the device directory.

Thereafter, when the file system receives a request from the I/O system, it simply calls the provided routines in the device driver to fulfill the request.

**RTFMT**

The RT-11 standard defines a peculiar software interleave and track-to-track skew as part of the format. The *rtFmt* parameter passed to **rt11FsDevInit()** should be **TRUE** if this formatting is desired. This should be the case if strict RT-11 compatibility is desired, or if files must be transferred between the development and target machines using the VxWorks-supplied RT-11 tools. Software interleave and skew will automatically be dealt with by **rt11FsLib**.

When *rtFmt* has been passed as **TRUE** and the maximum number of files is specified **RT\_FILES\_FOR\_2\_BLOCK\_SEG**, the driver does not need to do anything else to maintain RT-11 compatibility (except to add the track offset as described above).

Note that if the number of files specified is different than **RT\_FILES\_FOR\_2\_BLOCK\_SEG** under either a VxWorks system or an RT-11 system, compatibility is lost because VxWorks allocates a contiguous directory, whereas RT-11 systems create chained directories.

**MULTIPLE LOGICAL DEVICES AND RT-11 COMPATIBILITY**

The sector number passed to the sector read and write routines is an absolute number, starting from sector 0 at the beginning of the device. If desired, the driver may add an offset from the beginning of the physical device before the start of the logical device. This would normally be done by keeping an offset parameter in the device-specific structure of the driver, and adding the proper number of sectors to the sector number passed to the read and write routines.

The RT-11 standard defines the disk to start on track 1. Track 0 is set aside for boot information. Therefore, in order to retain true compatibility with RT-11 systems, a one-track offset (*i.e.*, the number of sectors in one track) needs to be added to the sector numbers passed to the sector read and write routines, and the device size needs to be declared as one track smaller than it actually is. This must be done by the driver using **rt11FsLib**; the library does not add such an offset automatically.

In the VxWorks RT-11 implementation, the directory is a fixed size, able to contain at least as many files as specified in the call to **rt11FsDevInit()**. If the maximum number of files is specified to be **RT\_FILES\_FOR\_2\_BLOCK\_SEG**, strict RT-11 compatibility is maintained, because this is the initial allocation in the RT-11 standard.

**RT-11 FILE NAMES** File names in the RT-11 file system use six characters, followed by a period (.), followed by an optional three-character extension.

**DIRECTORY ENTRIES**

An **ioctl()** call with the **FIODIRENTRY** function returns information about a particular

directory entry. A pointer to a `REQ_DIR_ENTRY` structure is passed as the parameter. The field `entryNum` in the `REQ_DIR_ENTRY` structure must be set to the desired entry number. The name of the file, its size (in bytes), and its creation date are returned in the structure. If the specified entry is empty (*i.e.*, if it represents an unallocated section of the disk), the name will be an empty string, the size will be the size of the available disk section, and the date will be meaningless. Typically, the entries are accessed sequentially, starting with `entryNum = 0`, until the terminating entry is reached, indicated by a return code of `ERROR`.

#### DIRECTORIES IN MEMORY

A copy of the directory for each volume is kept in memory (in the `RT_VOL_DESC` structure). This speeds up directory accesses, but requires that `rt11FsLib` be notified when disks are changed (*i.e.*, floppies are swapped). If the driver can find this out (by interrogating controller status or by receiving an interrupt), the driver simply calls `rt11FsReadyChange()` when a disk is inserted or removed. The library `rt11FsLib` will automatically try to remount the device next time it needs it.

If the driver does not have access to the information that disk volumes have been changed, the `changeNoWarn` parameter should be set to `TRUE` when the device is defined using `rt11FsDevInit()`. This will cause the disk to be automatically remounted before each `open()`, `creat()`, `delete()`, and directory listing.

The routine `rt11FsReadyChange()` can also be called by user tasks, by issuing an `ioctl()` call with `FIODISKCHANGE` as the function code.

#### ACCESSING THE RAW DISK

As a special case in `open()` and `creat()` calls, `rt11FsLib` recognizes a `NULL` file name to indicate access to the entire "raw" disk, as opposed to a file on the disk. Access in raw mode is useful for a disk that has no file system. For example, to initialize a new file system on the disk, use an `ioctl()` call with `FIODISKINIT`. To read the directory of a disk for which no file names are known, open the raw disk and use an `ioctl()` call with the function `FIODIRENTRY`.

#### HINTS

The RT-11 file system is much simpler than the more common UNIX or MS-DOS file systems. The advantage of RT-11 is its speed; file access is made in at most one seek because all files are contiguous. Some of the most common errors for users with a UNIX background are:

Only a single create at a time may be active per device.

File size is set by the first create and close sequence; use `lseek()` to ensure a specific file size; there is no append function to expand a file.

Files are strictly block oriented; unused portions of a block are filled with `NULLs` -- there is no end-of-file marker other than the last block.

**IOCTL FUNCTIONS** The rt11Fs file system supports the following `ioctl()` functions. The functions listed are defined in the header `ioLib.h`. Unless stated otherwise, the file descriptor used for these functions can be any file descriptor open to a file or to the volume itself.

#### **FIODISKFORMAT**

Formats the entire disk with appropriate hardware track and sector marks. No file system is initialized on the disk by this request. Note that this is a driver-provided function:

```
fd = open ("DEV1:", O_WRONLY);
status = ioctl (fd, FIODISKFORMAT, 0);
```

#### **FIODISKINIT**

Initializes an rt11Fs file system on the disk volume. This routine does not format the disk; formatting must be done by the driver. The file descriptor should be obtained by opening the entire volume in raw mode:

```
fd = open ("DEV1:", O_WRONLY);
status = ioctl (fd, FIODISKINIT, 0);
```

#### **FIODISKCHANGE**

Announces a media change. It performs the same function as `rt11FsReadyChange()`. This function may be called from interrupt level:

```
status = ioctl (fd, FIODISKCHANGE, 0);
```

#### **FIOGETNAME**

Gets the file name of the file descriptor and copies it to the buffer `nameBuf`:

```
status = ioctl (fd, FIOGETNAME, &nameBuf);
```

#### **FIORENAME**

Renames the file to the string `newname`:

```
status = ioctl (fd, FIORENAME, "newname");
```

#### **FIONREAD**

Copies to `unreadCount` the number of unread bytes in the file:

```
status = ioctl (fd, FIONREAD, &unreadCount);
```

#### **FIOFLUSH**

Flushes the file output buffer. It guarantees that any output that has been requested is actually written to the device.

```
status = ioctl (fd, FIOFLUSH, 0);
```

#### **FIOSEEK**

Sets the current byte offset in the file to the position specified by `newOffset`:

```
status = ioctl (fd, FIOSEEK, newOffset);
```

#### **FIOWHERE**

Returns the current byte position in the file. This is the byte offset of the next byte to

be read or written. It takes no additional argument:

```
position = ioctl (fd, FIOWHERE, 0);
```

#### **FIOSQUEEZE**

Coalesces fragmented free space on an rt11Fs volume:

```
status = ioctl (fd, FIOSQUEEZE, 0);
```

#### **FIODIRENTRY**

Copies information about the specified directory entries to a **REQ\_DIR\_ENTRY** structure that is defined in **ioLib.h**. The argument *req* is a pointer to a **REQ\_DIR\_ENTRY** structure. On entry, the structure contains the number of the directory entry for which information is requested. On return, the structure contains the information on the requested entry. For example, after the following:

```
REQ_DIR_ENTRY req;  
req.entryNum = 0;  
status = ioctl (fd, FIODIRENTRY, &req);
```

the request structure contains the name, size, and creation date of the file in the first entry (0) of the directory.

#### **FIOREADDIR**

Reads the next directory entry. The argument *dirStruct* is a DIR directory descriptor. Normally, **readdir()** is used to read a directory, rather than using the **FIOREADDIR** function directly. See **dirLib**.

```
DIR dirStruct;  
fd = open ("directory", O_RDONLY);  
status = ioctl (fd, FIOREADDIR, &dirStruct);
```

#### **FIOFSTATGET**

Gets file status information (directory entry data). The argument *statStruct* is a pointer to a stat structure that is filled with data describing the specified file. Normally, the **stat()** or **fstat()** routine is used to obtain file information, rather than using the **FIOFSTATGET** function directly. See **dirLib**.

```
struct stat statStruct;  
fd = open ("file", O_RDONLY);  
status = ioctl (fd, FIOFSTATGET, &statStruct);
```

Any other **ioctl()** function codes are passed to the block device driver for handling.

**INCLUDE FILES**    **rt11FsLib.h**

**SEE ALSO**        **ioLib**, **iosLib**, **ramDrv**, *VxWorks Programmer's Guide: I/O System, Local File Systems*

---

## schedPxBLib

**NAME** schedPxBLib – scheduling library (POSIX)

**ROUTINES** sched\_setparam() - set a task's priority (POSIX)  
sched\_getparam() - get the scheduling parameters for a specified task (POSIX)  
sched\_setscheduler() - set scheduling policy and scheduling parameters (POSIX)  
sched\_getscheduler() - get the current scheduling policy (POSIX)  
sched\_yield() - relinquish the CPU (POSIX)  
sched\_get\_priority\_max() - get the maximum priority (POSIX)  
sched\_get\_priority\_min() - get the minimum priority (POSIX)  
sched\_rr\_get\_interval() - get the current time slice (POSIX)

**DESCRIPTION** This library provides POSIX-compliance scheduling routines. The routines in this library allow the user to get and set priorities and scheduling schemes, get maximum and minimum priority values, and get the time slice if round-robin scheduling is enabled.

The POSIX standard specifies a priority numbering scheme in which higher priorities are indicated by larger numbers. The VxWorks native numbering scheme is the reverse of this, with higher priorities indicated by smaller numbers. For example, in the VxWorks native priority numbering scheme, the highest priority task has a priority of 0.

In VxWorks, POSIX scheduling interfaces are implemented using the POSIX priority numbering scheme. This means that the priority numbers used by this library *do not* match those reported and used in all the other VxWorks components. It is possible to change the priority numbering scheme used by this library by setting the global variable **posixPriorityNumbering**. If this variable is set to **FALSE**, the VxWorks native numbering scheme (small number = high priority) is used, and priority numbers used by this library will match those used by the other portions of VxWorks.

The routines in this library are compliant with POSIX 1003.1b. In particular, task priorities are set and reported through the structure **sched\_setparam**, which has a single member:

```
struct sched_param          /* Scheduling parameter structure */
{
    int sched_priority;      /* scheduling priority */
};
```

POSIX 1003.1b specifies this indirection to permit future extensions through the same calling interface. For example, because **sched\_setparam()** takes this structure as an argument (rather than using the priority value directly) its type signature need not change if future schedulers require other parameters.

**INCLUDE FILES** sched.h

**SEE ALSO** POSIX 1003.1b document, **taskLib**

## scsi1Lib

**NAME**            **scsi1Lib** – Small Computer System Interface (SCSI) library (SCSI-1)

**ROUTINES**        No Callable Routines

**DESCRIPTION**    This library implements the Small Computer System Interface (SCSI) protocol in a controller-independent manner. It implements only the SCSI initiator function; the library does not support a VxWorks target acting as a SCSI target. Furthermore, in the current implementation, a VxWorks target is assumed to be the only initiator on the SCSI bus, although there may be multiple targets (SCSI peripherals) on the bus.

The implementation is transaction based. A transaction is defined as the selection of a SCSI device by the initiator, the issuance of a SCSI command, and the sequence of data, status, and message phases necessary to perform the command. A transaction normally completes with a “Command Complete” message from the target, followed by disconnection from the SCSI bus. If the status from the target is “Check Condition,” the transaction continues; the initiator issues a “Request Sense” command to gain more information on the exception condition reported.

Many of the subroutines in **scsi1Lib** facilitate the transaction of frequently used SCSI commands. Individual command fields are passed as arguments from which SCSI Command Descriptor Blocks are constructed, and fields of a **SCSI\_TRANSACTION** structure are filled in appropriately. This structure, along with the **SCSI\_PHYS\_DEV** structure associated with the target SCSI device, is passed to the routine whose address is indicated by the **scsiTransact** field of the **SCSI\_CTRL** structure associated with the relevant SCSI controller.

The function variable **scsiTransact** is set by the individual SCSI controller driver. For off-board SCSI controllers, this routine rearranges the fields of the **SCSI\_TRANSACTION** structure into the appropriate structure for the specified hardware, which then carries out the transaction through firmware control. Drivers for an on-board SCSI-controller chip can use the **scsiTransact()** routine in **scsi1Lib** (which invokes the **scsi1Transact()** routine in **scsi1Lib**), as long as they provide the other functions specified in the **SCSI\_CTRL** structure.

Note that no disconnect/reconnect capability is currently supported.

### SUPPORTED SCSI DEVICES

The **scsi1Lib** library supports use of SCSI peripherals conforming to the standards specified in *Common Command Set (CCS) of the SCSI, Rev. 4.B*. Most SCSI peripherals currently offered support CCS. While an attempt has been made to have **scsi1Lib** support non-CCS peripherals, not all commands or features of this library are guaranteed to work with them. For example, auto-configuration may be impossible with non-CCS devices, if they do not support the INQUIRY command.

Not all classes of SCSI devices are supported. However, the **scsiLib** library provides the capability to transact any SCSI command on any SCSI device through the **FIOSCSICOMMAND** function of the **scsiIoctl()** routine.

Only direct-access devices (disks) are supported by a file system. For other types of devices, additional, higher-level software is necessary to map user-level commands to SCSI transactions.

### CONFIGURING SCSI CONTROLLERS

The routines to create and initialize a specific SCSI controller are particular to the controller and normally are found in its library module. The normal calling sequence is:

```
xxCtrlCreate (...); /* parameters are controller specific */
xxCtrlInit (...); /* parameters are controller specific */
```

The conceptual difference between the two routines is that **xxCtrlCreate()** alloc's memory for the **xx SCSI\_CTRL** data structure and initializes information that is never expected to change (for example, clock rate). The remaining fields in the **xx SCSI\_CTRL** structure are initialized by **xxCtrlInit()** and any necessary registers are written on the SCSI controller to effect the desired initialization. This routine can be called multiple times, although this is rarely required. For example, the bus ID of the SCSI controller can be changed without rebooting the VxWorks system.

### CONFIGURING PHYSICAL SCSI DEVICES

Before a device can be used, it must be "created," that is, declared. This is done with **scsiPhysDevCreate()** and can only be done after a **SCSI\_CTRL** structure exists and has been properly initialized.

```
SCSI_PHYS_DEV *scsiPhysDevCreate
(SCSI_CTRL * pScsiCtrl, /* ptr to SCSI controller info */
int devBusId, /* device's SCSI bus ID */
int devLUN, /* device's logical unit number */
int reqSenseLength, /* length of REQUEST SENSE data dev returns */
int devType, /* type of SCSI device */
BOOL removable, /* whether medium is removable */
int numBlocks, /* number of blocks on device */
int blockSize /* size of a block in bytes */
)
```

Several of these parameters can be left unspecified, as follows:

*reqSenseLength*

If 0, issue a **REQUEST\_SENSE** to determine a request sense length.

*devType*

If -1, issue an **INQUIRY** to determine the device type.

*numBlocks, blockSize*

If 0, issue a **READ\_CAPACITY** to determine the number of blocks.

The above values are recommended, unless the device does not support the required commands, or other non-standard conditions prevail.

#### LOGICAL PARTITIONS ON BLOCK DEVICES

It is possible to have more than one logical partition on a SCSI block device. This capability is currently not supported for removable media devices. A partition is an array of contiguously addressed blocks with a specified starting block address and a specified number of blocks. The **scsiBlkDevCreate()** routine is called once for each block device partition. Under normal usage, logical partitions should not overlap.

```
SCSI_BLK_DEV *scsiBlkDevCreate
(
    SCSI_PHYS_DEV * pScsiPhysDev, /* ptr to SCSI physical device info */
    int             numBlocks,     /* number of blocks in block device */
    int             blockOffset    /* address of first block in volume */
)
```

Note that if *numBlocks* is 0, the rest of the device is used.

#### ATTACHING FILE SYSTEMS TO LOGICAL PARTITIONS

Files cannot be read or written to a disk partition until a file system (such as dosFs or rt11Fs) has been initialized on the partition. For more information, see the documentation in **dosFsLib** or **rt11FsLib**.

#### TRANSMITTING ARBITRARY COMMANDS TO SCSI DEVICES

The **scsi1Lib** library provides routines that implement many common SCSI commands. Still, there are situations that require commands that are not supported by **scsi1Lib** (for example, writing software to control non-direct access devices). Arbitrary commands are handled with the **FIOCSICOMMAND** option to **scsiIoctl()**. The *arg* parameter for **FIOCSICOMMAND** is a pointer to a valid **SCSI\_TRANSACTION** structure. Typically, a call to **scsiIoctl()** is written as a subroutine of the form:

```
STATUS myScsiCommand
(
    SCSI_PHYS_DEV * pScsiPhysDev, /* ptr to SCSI physical device */
    char *         buffer,        /* ptr to data buffer */
    int            bufLength,     /* length of buffer in bytes */
    int            someParam     /* param. specifiable in cmd block */
)
{
    SCSI_COMMAND myScsiCmdBlock; /* SCSI command byte array */
    SCSI_TRANSACTION myScsiXaction; /* info on a SCSI transaction */
    /* fill in fields of SCSI_COMMAND structure */
    myScsiCmdBlock [0] = MY_COMMAND_OPCODE; /* the required opcode */
}
```

```

.
myScsiCmdBlock [X] = (UINT8) someParam;      /* for example */
.
myScsiCmdBlock [N-1] = MY_CONTROL_BYTE;     /* typically == 0 */
/* fill in fields of SCSI_TRANSACTION structure */
myScsiXaction.cmdAddress    = myScsiCmdBlock;
myScsiXaction.cmdLength    = <# of valid bytes in myScsiCmdBlock>;
myScsiXaction.dataAddress  = (UINT8 *) buffer;
myScsiXaction.dataDirection = <O_RDONLY (0) or O_WRONLY (1)>;
myScsiXaction.dataLength   = bufLength;
myScsiXaction.cmdTimeout   = timeout in usec;
/* if dataDirection is O_RDONLY, and the length of the input data is
 * variable, the following parameter specifies the byte # (min == 0)
 * of the input data which will specify the additional number of
 * bytes available
 */
myScsiXaction.addLengthByte = X;
if (scsiIoctl (pScsiPhysDev, FIOSCSICOMMAND, &myScsiXaction) == OK)
    return (OK);
else
    /* optionally perform retry or other action based on value of
     * myScsiXaction.statusByte
     */
    return (ERROR);
}

```

**INCLUDE FILES**    scsiLib.h, scsi1Lib.h

**SEE ALSO**        **dosFsLib**, **rt11FsLib**, *American National Standards for Information Systems - Small Computer System Interface (SCSI)*, *ANSI X3.131-1986*, *VxWorks Programmer's Guide: I/O System, Local File Systems*

---

## scsi2Lib

**NAME**            scsi2Lib – Small Computer System Interface (SCSI) library (SCSI-2)

**ROUTINES**

- scsi2IfInit()** - initialize the SCSI-2 interface to **scsiLib**
- scsiTargetOptionsSet()** - set options for one or all SCSI targets
- scsiTargetOptionsGet()** - get options for one or all SCSI targets
- scsiTargetOptionsShow()** - display options for specified SCSI target
- scsiPhysDevShow()** - show status information for a physical device
- scsiCacheSynchronize()** - synchronize the caches for data coherency

**scsi2Lib**

**scsiIdentMsgBuild()** - build an identification message  
**scsiIdentMsgParse()** - parse an identification message  
**scsiMsgOutComplete()** - perform post-processing after a SCSI message is sent  
**scsiMsgOutReject()** - perform post-processing when an outgoing message is rejected  
**scsiMsgInComplete()** - handle a complete SCSI message received from the target  
**scsiSyncXferNegotiate()** - initiate or continue negotiating transfer parameters  
**scsiWideXferNegotiate()** - initiate or continue negotiating wide parameters  
**scsiThreadInit()** - perform generic SCSI thread initialization  
**scsiCacheSnoopEnable()** - inform SCSI that hardware snooping of caches is enabled  
**scsiCacheSnoopDisable()** - inform SCSI that hardware snooping of caches is disabled

**DESCRIPTION**

This library implements the Small Computer System Interface (SCSI) protocol in a controller-independent manner. It implements only the SCSI initiator function as defined in the SCSI-2 ANSI specification. This library does not support a VxWorks target acting as a SCSI target.

The implementation is transaction based. A transaction is defined as the selection of a SCSI device by the initiator, the issuance of a SCSI command, and the sequence of data, status, and message phases necessary to perform the command. A transaction normally completes with a "Command Complete" message from the target, followed by disconnection from the SCSI bus. If the status from the target is "Check Condition," the transaction continues; the initiator issues a "Request Sense" command to gain more information on the exception condition reported.

Many of the subroutines in **scsi2Lib** facilitate the transaction of frequently used SCSI commands. Individual command fields are passed as arguments from which SCSI Command Descriptor Blocks are constructed, and fields of a **SCSI\_TRANSACTION** structure are filled in appropriately. This structure, along with the **SCSI\_PHYS\_DEV** structure associated with the target SCSI device, is passed to the routine whose address is indicated by the **scsiTransact** field of the **SCSI\_CTRL** structure associated with the relevant SCSI controller. The above mentioned structures are defined in **scsi2Lib.h**.

The function variable **scsiTransact** is set by the individual SCSI controller driver. For off-board SCSI controllers, this routine rearranges the fields of the **SCSI\_TRANSACTION** structure into the appropriate structure for the specified hardware, which then carries out the transaction through firmware control. Drivers for an on-board SCSI-controller chip can use the **scsiTransact()** routine in **scsiLib** (which invokes the **scsi2Transact()** routine in **scsi2Lib**), as long as they provide the other functions specified in the **SCSI\_CTRL** structure.

**SCSI TRANSACTION TIMEOUT**

Associated with each transaction is a time limit (specified in microseconds, but measured with the resolution of the system clock). If the transaction has not completed within this time limit, the SCSI library aborts it; the called routine fails with a corresponding error code. The timeout period includes time spent waiting for the target device to become free to accept the command.

The semantics of the timeout should guarantee that the caller waits no longer than the transaction timeout period, but in practice this may depend on the state of the SCSI bus and the connected target device when the timeout occurs. If the target behaves correctly according to the SCSI specification, proper timeout behavior results. However, in certain unusual cases—for example, when the target does not respond to an asserted ATN signal—the caller may remain blocked for longer than the timeout period.

If the transaction timeout causes problems in your system, you can set the value of either or both the global variables “scsi{Min,Max}Timeout”. These specify (in microseconds) the global minimum and maximum timeout periods, which override (clip) the value specified for a transaction. They may be changed at any time and affect all transactions issued after the new values are set. The range of both these variable is 0 to 0xffffffff (zero to about 4295 seconds).

#### SCSI TRANSACTION PRIORITY

Each transaction also has an associated priority used by the SCSI library when selecting the next command to issue when the SCSI system is idle. It chooses the highest priority transaction that can be dispatched on an available physical device. If there are several equal-priority transactions available, the SCSI library uses a simple round-robin scheme to avoid favoring the same physical device.

Priorities range from 0 (highest) to 255 (lowest), which is the same as task priorities. The priority `SCSI_THREAD_TASK_PRIORITY` can be used to give the transaction the same priority as the calling task (this is the method used internally by this SCSI-2 library).

#### SUPPORTED SCSI DEVICES

This library requires peripherals that conform to the SCSI-2 ANSI standard; in particular, the INQUIRY, REQUEST SENSE, and TEST UNIT READY commands must be supported as specified by this standard. In general, the SCSI library is self-configuring to work with any device that meets these requirements.

Peripherals that support identification and the SCSI message protocol are strongly recommended as these provide maximum performance.

In theory, all classes of SCSI devices are supported. The **scsiLib** library provides the capability to transact any SCSI command on any SCSI device through the `FIOSCSI` function of the `scsiIoctl()` routine (which invokes the `scsi2Ioctl()` routine in **scsi2Lib**).

Only direct-access devices (disks) are supported by file systems like `dosFs`, `rt11Fs` and `rawFs`. These file systems employ routines in **scsiDirectLib** (most of which are described in **scsiLib** but defined in **scsiDirectLib**). In the case of sequential-access devices (tapes), higher-level tape file systems, like `tapeFs`, make use of **scsiSeqLib**. For other types of devices, additional, higher-level software is necessary to map user-level commands to SCSI transactions.

#### DISCONNECT/RECONNECT SUPPORT

The target device can be disconnected from the SCSI bus while it carries out a SCSI

command; in this way, commands to multiple SCSI devices can be overlapped to improve overall SCSI throughput. There are no restrictions on the number of pending, disconnected commands or the order in which they are resumed. The SCSI library serializes access to the device according to the capabilities and status of the device (see the following section).

Use of the disconnect/reconnect mechanism is invisible to users of the SCSI library. It can be enabled and disabled separately for each target device (see **scsiTargetOptionsSet()**). Note that support for disconnect/reconnect depends on the capabilities of the controller and its driver (see below).

#### **TAGGED COMMAND QUEUEING SUPPORT**

If the target device conforms to the ANSI SCSI-2 standard and indicates (using the INQUIRY command) that it supports command queuing, the SCSI library allows new commands to be started on the device whenever the SCSI bus is idle. That is, it executes multiple commands concurrently on the target device. By default, commands are tagged with a SIMPLE QUEUE TAG message. Up to 256 commands can be executing concurrently.

The SCSI library correctly handles contingent allegiance conditions that arise while a device is executing tagged commands. (A contingent allegiance condition exists when a target device is maintaining sense data that the initiator should use to correctly recover from an error condition.) It issues an untagged REQUEST SENSE command, and stops issuing tagged commands until the sense recovery command has completed.

For devices that do not support command queuing, the SCSI library only issues a new command when the previous one has completed. These devices can only execute a single command at once.

Use of tagged command queuing is normally invisible to users of the SCSI library. If necessary, the default tag type and maximum number of tags may be changed on a per-target basis, using **scsiTargetOptionsSet()**.

#### **SYNCHRONOUS TRANSFER PROTOCOL SUPPORT**

If the SCSI controller hardware supports the synchronous transfer protocol, **scsiLib** negotiates with the target device to determine whether to use synchronous or asynchronous transfers. Either VxWorks or the target device may start a round of negotiation. Depending on the controller hardware, synchronous transfer rates up to the maximum allowed by the SCSI-2 standard (10 Mtransfers/second) can be used.

Again, this is normally invisible to users of the SCSI library, but synchronous transfer parameters may be set or disabled on a per-target basis by using **scsiTargetOptionsSet()**.

#### **WIDE DATA TRANSFER SUPPORT**

If the SCSI controller supports the wide data transfer protocol, **scsiLib** negotiates wide data transfer parameters with the target device, if that device also supports wide transfers. Either VxWorks or the target device may start a round of negotiation. Wide data transfer parameters are negotiated prior to the synchronous data transfer parameters, as specified

by the SCSI-2 ANSI specification. In conjunction with synchronous transfer, up to a maximum of 20MB/sec. can be attained.

Wide data transfer negotiation is invisible to users of this library, but it is possible to enable or disable wide data transfers and the parameters on a per-target basis by using `scsiTargetOptionsSet()`.

**SCSI BUS RESET** The SCSI library implements the ANSI “hard reset” option. Any transactions in progress when a SCSI bus reset is detected fail with an error code indicating termination due to bus reset. Any transactions waiting to start executing are then started normally.

#### CONFIGURING SCSI CONTROLLERS

The routines to create and initialize a specific SCSI controller are particular to the controller and normally are found in its library module. The normal calling sequence is:

```
xxCtrlCreate (...); /* parameters are controller specific */
xxCtrlInit (...); /* parameters are controller specific */
```

The conceptual difference between the two routines is that `xxCtrlCreate()` alloc’s memory for the `xx_SCSCI_CTRL` data structure and initializes information that is never expected to change (for example, clock rate). The remaining fields in the `xx_SCSCI_CTRL` structure are initialized by `xxCtrlInit()` and any necessary registers are written on the SCSI controller to effect the desired initialization. This routine can be called multiple times, although this is rarely required. For example, the bus ID of the SCSI controller can be changed without rebooting the VxWorks system.

#### CONFIGURING PHYSICAL SCSI DEVICES

Before a device can be used, it must be “created,” that is, declared. This is done with `scsiPhysDevCreate()` and can only be done after a `SCSCI_CTRL` structure exists and has been properly initialized.

```
SCSI_PHYS_DEV *scsiPhysDevCreate
(
    SCSCI_CTRL * pScsiCtrl, /* ptr to SCSI controller info */
    int devBusId,          /* device's SCSI bus ID */
    int devLUN,            /* device's logical unit number */
    int reqSenseLength,    /* length of REQUEST_SENSE data dev returns */
    int devType,           /* type of SCSI device */
    BOOL removable,        /* whether medium is removable */
    int numBlocks,         /* number of blocks on device */
    int blockSize          /* size of a block in bytes */
)
```

Several of these parameters can be left unspecified, as follows:

*reqSenseLength*

If 0, issue a `REQUEST_SENSE` to determine a request sense length.

*devType*

This parameter is ignored: an INQUIRY command is used to ascertain the device type. A value of NONE (-1) is the recommended placeholder.

*numBlocks, blockSize*

If 0, issue a **READ\_CAPACITY** to determine the number of blocks.

The above values are recommended, unless the device does not support the required commands, or other non-standard conditions prevail.

#### LOGICAL PARTITIONS ON DIRECT-ACCESS BLOCK DEVICES

It is possible to have more than one logical partition on a SCSI block device. This capability is currently not supported for removable media devices. A partition is an array of contiguously addressed blocks with a specified starting block address and specified number of blocks. The **scsiBlkDevCreate()** routine is called once for each block device partition. Under normal usage, logical partitions should not overlap.

```
SCSI_BLK_DEV *scsiBlkDevCreate
(
    SCSI_PHYS_DEV * pScsiPhysDev, /* ptr to SCSI physical device info */
    int             numBlocks,     /* number of blocks in block device */
    int             blockOffset    /* address of first block in volume */
)
```

Note that if *numBlocks* is 0, the rest of the device is used.

#### ATTACHING DISK FILE SYSTEMS TO LOGICAL PARTITIONS

Files cannot be read or written to a disk partition until a file system (for example, dosFs, rt11Fs, or rawFs) has been initialized on the partition. For more information, see the relevant documentation in **dosFsLib**, **rt11FsLib**, or **rawFsLib**.

#### USING A SEQUENTIAL-ACCESS BLOCK DEVICE

The entire volume (tape) on a sequential-access block device is treated as a single raw file. This raw file is made available to higher-level layers like tapeFs by the **scsiSeqDevCreate()** routine, described in **scsiSeqLib**. The **scsiSeqDevCreate()** routine is called once for a given SCSI physical device.

```
SEQ_DEV *scsiSeqDevCreate
(
    SCSI_PHYS_DEV *pScsiPhysDev /* ptr to SCSI physical device info */
)
```

#### TRANSMITTING ARBITRARY COMMANDS TO SCSI DEVICES

The **scsi2Lib**, **scsiCommonLib**, **scsiDirectLib**, and **scsiSeqLib** libraries collectively provide routines that implement all mandatory SCSI-2 direct-access and sequential-access commands. Still, there are situations that require commands not supported by these libraries (for example, writing software that needs to use an optional SCSI-2 command).

Arbitrary commands are handled with the FIOSCSICOMMAND option to `scsiIoctl()`. The `arg` parameter for FIOSCSICOMMAND is a pointer to a valid SCSI\_TRANSACTION structure. Typically, a call to `scsiIoctl()` is written as a routine of the form:

```

STATUS myScsiCommand
(
    SCSI_PHYS_DEV * pScsiPhysDev, /* ptr to SCSI physical device */
    char *         buffer,        /* ptr to data buffer */
    int            bufLength,     /* length of buffer in bytes */
    int            someParam      /* param. specifiabile in cmd block */
)
{
    SCSI_COMMAND myScsiCmdBlock; /* SCSI command byte array */
    SCSI_TRANSACTION myScsiXaction; /* info on a SCSI transaction */
    /* fill in fields of SCSI_COMMAND structure */
    myScsiCmdBlock [0] = MY_COMMAND_OPCODE; /* the required opcode */
    .
    myScsiCmdBlock [X] = (UINT8) someParam; /* for example */
    .
    myScsiCmdBlock [N-1] = MY_CONTROL_BYTE; /* typically == 0 */
    /* fill in fields of SCSI_TRANSACTION structure */
    myScsiXaction.cmdAddress = myScsiCmdBlock;
    myScsiXaction.cmdLength = <# of valid bytes in myScsiCmdBlock>;
    myScsiXaction.dataAddress = (UINT8 *) buffer;
    myScsiXaction.dataDirection = <O_RDONLY (0) or O_WRONLY (1)>;
    myScsiXaction.dataLength = bufLength;
    myScsiXaction.addLengthByte = 0; /* no longer used */
    myScsiXaction.cmdTimeout = <timeout in usec>;
    myScsiXaction.tagType = SCSI_TAG_{DEFAULT, UNTAGGED,
                                     SIMPLE, ORDERED, HEAD_OF_Q};
    myScsiXaction.priority = [ 0 (highest) to 255 (lowest) ];
    if (scsiIoctl (pScsiPhysDev, FIOSCSICOMMAND, &myScsiXaction) == OK)
        return (OK);
    else
        /* optionally perform retry or other action based on value of
         * myScsiXaction.statusByte
         */
        return (ERROR);
}

```

INCLUDE FILES    `scsiLib.h`, `scsi2Lib.h`

SEE ALSO        `dosFsLib`, `rt11FsLib`, `rawFsLib`, `tapeFsLib`, `scsiLib`, `scsiCommonLib`, `scsiDirectLib`, `scsiSeqLib`, `scsiMgrLib`, `scsiCtrlLib`, *American National Standard for Information Systems - Small Computer System Interface (SCSI-2)*, *ANSI X3T9*, *VxWorks Programmer's Guide: I/O System*, *Local File Systems*

## scsiCommonLib

<b>NAME</b>	<b>scsiCommonLib</b> – SCSI library common commands for all devices (SCSI-2)								
<b>ROUTINES</b>	No Callable Routines.								
<b>DESCRIPTION</b>	<p>This library contains commands common to all SCSI devices. The content of this library is separated from the other SCSI libraries in order to create an additional layer for better support of all SCSI devices.</p> <p>Commands in this library include:</p> <table><thead><tr><th>Command</th><th>Op Code</th></tr></thead><tbody><tr><td>INQUIRY</td><td>(0x12)</td></tr><tr><td>REQUEST SENSE</td><td>(0x03)</td></tr><tr><td>TEST UNIT READY</td><td>(0x00)</td></tr></tbody></table>	Command	Op Code	INQUIRY	(0x12)	REQUEST SENSE	(0x03)	TEST UNIT READY	(0x00)
Command	Op Code								
INQUIRY	(0x12)								
REQUEST SENSE	(0x03)								
TEST UNIT READY	(0x00)								
<b>INCLUDE FILES</b>	<b>scsiLib.h, scsi2Lib.h</b>								
<b>SEE ALSO</b>	<b>dosFsLib, rt11FsLib, rawFsLib, tapeFsLib, scsi2Lib</b> , <i>VxWorks Programmer's Guide: I/O System, Local File Systems</i>								

---

## scsiCtrlLib

<b>NAME</b>	<b>scsiCtrlLib</b> – SCSI thread-level controller library (SCSI-2)
<b>ROUTINES</b>	No Callable Routines.
<b>DESCRIPTION</b>	<p>The purpose of the SCSI controller library is to support basic SCSI controller drivers that rely on a higher level of software in order to manage SCSI transactions. More advanced SCSI I/O processors do not require this protocol engine since software support for SCSI transactions is provided at the SCSI I/O processor level.</p> <p>This library provides all the high-level routines that manage the state of the SCSI threads and guide the SCSI I/O transaction through its various stages:</p> <ul style="list-style-type: none"><li>– selecting a SCSI peripheral device;</li><li>– sending the identify message in order to establish the ITL nexus;</li><li>– cycling through information transfer, message and data, and status phases;</li><li>– handling bus-initiated reselects.</li></ul>

The various stages of the SCSI I/O transaction are reported to the SCSI manager as SCSI events. Event selection and management is handled by routines in this library.

**INCLUDE FILES**    **scsiLib.h, scsi2Lib.h**

**SEE ALSO**        **scsiLib, scsi2Lib, scsiCommonLib, scsiDirectLib, scsiSeqLib, scsiMgrLib**, *American National Standard for Information Systems - Small Computer System Interface (SCSI-2), ANSI X3T9, VxWorks Programmer's Guide: I/O System, Local File Systems*

---

## scsiDirectLib

**NAME**            **scsiDirectLib** – SCSI library for direct access devices (SCSI-2)

**ROUTINES**      **scsiStartStopUnit()** - issue a START\_STOP\_UNIT command to a SCSI device  
**scsiReserve()**    - issue a RESERVE command to a SCSI device  
**scsiRelease()**    - issue a RELEASE command to a SCSI device

**DESCRIPTION**    This library contains commands common to all direct-access SCSI devices. These routines are separated from **scsi2Lib** in order to create an additional layer for better support of all SCSI direct-access devices.

Commands in this library include:

<b>Command</b>	<b>Op Code</b>
FORMAT UNIT	(0x04)
READ (6)	(0x08)
READ (10)	(0x28)
READ CAPACITY	(0x25)
RELEASE	(0x17)
RESERVE	(0x16)
MODE SELECT (6)	(0x15)
MODE SELECT (10)	(0x55)
MODE SENSE (6)	(0x1a)
MODE SENSE (10)	(0x5a)
START STOP UNIT	(0x1b)
WRITE (6)	(0x0a)
WRITE (10)	(0x2a)

**INCLUDE FILES**    **scsiLib.h, scsi2Lib.h**

**SEE ALSO**        **dosFsLib, rt11FsLib, rawFsLib, scsi2Lib**, *VxWorks Programmer's Guide: I/O System, Local File Systems*

S

---

## scsiLib

<b>NAME</b>	<b>scsiLib</b> – Small Computer System Interface (SCSI) library
<b>ROUTINES</b>	<p><b>scsiPhysDevDelete()</b> - delete a SCSI physical-device structure</p> <p><b>scsiPhysDevCreate()</b> - create a SCSI physical device structure</p> <p><b>scsiPhysDevIdGet()</b> - return a pointer to a <b>SCSI_PHYS_DEV</b> structure</p> <p><b>scsiAutoConfig()</b> - configure all devices connected to a SCSI controller</p> <p><b>scsiShow()</b> - list the physical devices attached to a SCSI controller</p> <p><b>scsiBlkDevCreate()</b> - define a logical partition on a SCSI block device</p> <p><b>scsiBlkDevInit()</b> - initialize fields in a SCSI logical partition</p> <p><b>scsiBlkDevShow()</b> - show the <b>BLK_DEV</b> structures on a specified physical device</p> <p><b>scsiBusReset()</b> - pulse the reset signal on the SCSI bus</p> <p><b>scsiIoctl()</b> - perform a device-specific I/O control function</p> <p><b>scsiFormatUnit()</b> - issue a <b>FORMAT_UNIT</b> command to a SCSI device</p> <p><b>scsiModeSelect()</b> - issue a <b>MODE_SELECT</b> command to a SCSI device</p> <p><b>scsiModeSense()</b> - issue a <b>MODE_SENSE</b> command to a SCSI device</p> <p><b>scsiReadCapacity()</b> - issue a <b>READ_CAPACITY</b> command to a SCSI device</p> <p><b>scsiRdSecs()</b> - read sector(s) from a SCSI block device</p> <p><b>scsiWrtSecs()</b> - write sector(s) to a SCSI block device</p> <p><b>scsiTestUnitRdy()</b> - issue a <b>TEST_UNIT_READY</b> command to a SCSI device</p> <p><b>scsiInquiry()</b> - issue an <b>INQUIRY</b> command to a SCSI device</p> <p><b>scsiReqSense()</b> - issue a <b>REQUEST_SENSE</b> command to a SCSI device and read results</p>
<b>DESCRIPTION</b>	<p>The purpose of this library is to switch SCSI function calls (the common SCSI-1 and SCSI-2 calls listed above) to either <b>scsi1Lib</b> or <b>scsi2Lib</b>, depending upon the SCSI configuration in the Board Support Package (BSP). The normal usage is to configure SCSI-2. However, SCSI-1 is configured when device incompatibilities exist. VxWorks can be configured with either SCSI-1 or SCSI-2, but not both SCSI-1 and SCSI-2 simultaneously.</p> <p>For more information about SCSI-1 functionality, refer to <b>scsi1Lib</b>. For more information about SCSI-2, refer to <b>scsi2Lib</b>.</p>
<b>INCLUDE FILES</b>	<b>scsiLib.h</b> , <b>scsi1Lib.h</b> , <b>scsi2Lib.h</b>
<b>SEE ALSO</b>	<b>dosFsLib</b> , <b>rt11FsLib</b> , <b>rawFsLib</b> , <b>scsi1Lib</b> , <b>scsi2Lib</b> , <i>VxWorks Programmer's Guide: I/O System, Local File Systems</i>

---

## scsiMgrLib

<b>NAME</b>	<b>scsiMgrLib</b> – SCSI manager library (SCSI-2)
<b>ROUTINES</b>	<b>scsiMgrEventNotify()</b> - notify the SCSI manager of a SCSI (controller) event <b>scsiMgrBusReset()</b> - handle a controller-bus reset event <b>scsiMgrCtrlEvent()</b> - send an event to the SCSI controller state machine <b>scsiMgrThreadEvent()</b> - send an event to the thread state machine <b>scsiMgrShow()</b> - show status information for the SCSI manager
<b>DESCRIPTION</b>	<p>This SCSI-2 library implements the SCSI manager. The SCSI manager manages SCSI threads between requesting VxWorks tasks and the SCSI controller. The SCSI manager handles SCSI events and SCSI threads but allocation and de-allocation of SCSI threads is not the manager's responsibility. SCSI thread management includes despatching threads and scheduling multiple threads (which are performed by the SCSI manager, plus allocation and de-allocation of threads (which are performed by routines in <b>scsi2Lib</b>).</p> <p>The SCSI manager is spawned as a task on initialization of the SCSI interface within VxWorks. The entry point of the SCSI manager task is <b>scsiMgr()</b>. The task is usually spawned during initialization of the SCSI controller driver. The driver's <b>xxCtrlCreateScsi2()</b> routine is typically responsible for such SCSI interface initializations.</p> <p>Once the SCSI manager has been initialized, it is ready to handle SCSI requests from VxWorks tasks. The SCSI manager has the following responsibilities:</p> <ul style="list-style-type: none"><li>– It processes requests from client tasks.</li><li>– It activates a SCSI transaction thread by appending it to the target device's wait queue and allocating a specified time period to execute a transaction.</li><li>– It handles timeout events which cause threads to be aborted.</li><li>– It receives event notifications from the SCSI driver interrupt service routine (ISR) and processes the event.</li><li>– It responds to events generated by the controller hardware, such as disconnection and information transfer requests.</li><li>– It replies to clients when their requests have completed or aborted.</li></ul> <p>One SCSI manager task must be spawned per SCSI controller. Thus, if a particular hardware platform contains more than one SCSI controller then that number of SCSI manager tasks must be spawned by the controller-driver initialization routine.</p>
<b>INCLUDE FILES</b>	<b>scsiLib.h</b> , <b>scsi2Lib.h</b>
<b>SEE ALSO</b>	<b>scsiLib</b> , <b>scsi2Lib</b> , <b>scsiCommonLib</b> , <b>scsiDirectLib</b> , <b>scsiSeqLib</b> , <b>scsiCtrlLib</b> , <i>American National Standard for Information Systems - Small Computer System Interface (SCSI-2)</i> , <i>ANSI X3T9, VxWorks Programmer's Guide: I/O System, Local File Systems</i>

---

## scsiSeqLib

**NAME**            **scsiSeqLib** – SCSI sequential access device library (SCSI-2)

**ROUTINES**

- scsiSeqDevCreate()** - create a SCSI sequential device
- scsiErase()** - issue an **ERASE** command to a SCSI device
- scsiTapeModeSelect()** - issue a **MODE\_SELECT** command to a SCSI tape device
- scsiTapeModeSense()** - issue a **MODE\_SENSE** command to a SCSI tape device
- scsiSeqReadBlockLimits()** - issue a **READ\_BLOCK\_LIMITS** command to a SCSI device
- scsiRdTape()** - read bytes or blocks from a SCSI tape device
- scsiWrtTape()** - write data to a SCSI tape device
- scsiRewind()** - issue a **REWIND** command to a SCSI device
- scsiReserveUnit()** - issue a **RESERVE UNIT** command to a SCSI device
- scsiReleaseUnit()** - issue a **RELEASE UNIT** command to a SCSI device
- scsiLoadUnit()** - issue a **LOAD/UNLOAD** command to a SCSI device
- scsiWrtFileMarks()** - write file marks to a SCSI sequential device
- scsiSpace()** - move the tape on a specified physical SCSI device
- scsiSeqStatusCheck()** - detect a change in media
- scsiSeqIoctl()** - perform an I/O control function for sequential access devices

**DESCRIPTION**    This library contains commands common to all sequential-access SCSI devices. Such devices are usually SCSI tape devices. These routines are separated from **scsi2Lib** in order to create an additional layer for better support of all SCSI sequential devices.

SCSI commands in this library include:

<b>Command</b>	<b>Op Code</b>
ERASE	(0x19)
MODE SELECT (6)	(0x15)
MODE_SENSE (6)	(0x1a)
READ (6)	(0x08)
READ BLOCK LIMITS	(0x05)
RELEASE UNIT	(0x17)
RESERVE UNIT	(0x16)
REWIND	(0x01)
SPACE	(0x11)
WRITE (6)	(0x0a)
WRITE FILEMARKS	(0x10)
LOAD/UNLOAD	(0x1b)

The SCSI routines implemented here operate mostly on a **SCSI\_SEQ\_DEV** structure. This structure acts as an interface between this library and a higher-level layer. The **SEQ\_DEV** structure is analogous to the **BLK\_DEV** structure for block devices.

The `scsiSeqDevCreate()` routine creates a `SCSI_SEQ_DEV` structure whose first element is a `SEQ_DEV`, operated upon by higher layers. This routine publishes all functions to be invoked by higher layers and maintains some state information (for example, block size) for tracking SCSI-sequential-device information.

**INCLUDE FILES** `scsiLib.h`, `scsi2Lib.h`

**SEE ALSO** `tapeFsLib`, `scsi2Lib`, *VxWorks Programmer's Guide: I/O System, Local File Systems*

---

## selectLib

**NAME** `selectLib` – UNIX BSD 4.3 select library

**ROUTINES**

- `selectInit()` - initialize the select facility
- `select()` - pend on a set of file descriptors
- `selWakeup()` - wake up a task pending in `select()`
- `selWakeupAll()` - wake up all tasks in a `select()` wake-up list
- `selNodeAdd()` - add a wake-up node to a `select()` wake-up list
- `selNodeDelete()` - find and delete a node from a `select()` wake-up list
- `selWakeupListInit()` - initialize a `select()` wake-up list
- `selWakeupListTerm()` - terminate a `select()` wake-up list
- `selWakeupListLen()` - get the number of nodes in a `select()` wake-up list
- `selWakeupType()` - get the type of a `select()` wake-up node

**DESCRIPTION** This library provides a BSD 4.3 compatible *select* facility to wait for activity on a set of file descriptors. `selectLib` provides a mechanism that gives a driver the ability to detect pending tasks that are awaiting activity on the driver's device. This allows a driver's interrupt service routine to wake up such tasks directly, eliminating the need for polling.

Applications can use `select()` with pipes and serial devices, in addition to sockets. Also, `select()` examines *write* file descriptors in addition to *read* file descriptors; however, exception file descriptors remain unsupported.

Typically, application developers need concern themselves only with the `select()` call. However, driver developers should become familiar with the other routines that may be used with `select()`, if they wish to support the `select()` mechanism.

The select facility is included in a system when VxWorks is configured with the `INCLUDE_SELECT` component.

**INCLUDE FILES** `selectLib.h`

**SEE ALSO** *VxWorks Programmer's Guide: I/O System*

---

## semBLib

**NAME** semBLib – binary semaphore library

**ROUTINES** semBCreate() - create and initialize a binary semaphore

**DESCRIPTION** This library provides the interface to VxWorks binary semaphores. Binary semaphores are the most versatile, efficient, and conceptually simple type of semaphore. They can be used to: (1) control mutually exclusive access to shared devices or data structures, or (2) synchronize multiple tasks, or task-level and interrupt-level processes. Binary semaphores form the foundation of numerous VxWorks facilities.

A binary semaphore can be viewed as a cell in memory whose contents are in one of two states, full or empty. When a task takes a binary semaphore, using **semTake()**, subsequent action depends on the state of the semaphore:

- (1) If the semaphore is full, the semaphore is made empty, and the calling task continues executing.
- (2) If the semaphore is empty, the task will be blocked, pending the availability of the semaphore. If a timeout is specified and the timeout expires, the pended task will be removed from the queue of pended tasks and enter the ready state with an **ERROR** status. A pended task is ineligible for CPU allocation. Any number of tasks may be pended simultaneously on the same binary semaphore.

When a task gives a binary semaphore, using **semGive()**, the next available task in the pend queue is unblocked. If no task is pending on this semaphore, the semaphore becomes full. Note that if a semaphore is given, and a task is unblocked that is of higher priority than the task that called **semGive()**, the unblocked task will preempt the calling task.

### MUTUAL EXCLUSION

To use a binary semaphore as a means of mutual exclusion, first create it with an initial state of full. For example:

```
SEM_ID semMutex;  
/* create a binary semaphore that is initially full */  
semMutex = semBCreate (SEM_Q_PRIORITY, SEM_FULL);
```

Then guard a critical section or resource by taking the semaphore with **semTake()**, and exit the section or release the resource by giving the semaphore with **semGive()**. For example:

```
semTake (semMutex, WAIT_FOREVER);  
... /* critical region, accessible only by one task at a time */  
  
semGive (semMutex);
```

While there is no restriction on the same semaphore being given, taken, or flushed by multiple tasks, it is important to ensure the proper functionality of the mutual-exclusion construct. While there is no danger in any number of processes taking a semaphore, the giving of a semaphore should be more carefully controlled. If a semaphore is given by a task that did not take it, mutual exclusion could be lost.

## SYNCHRONIZATION

To use a binary semaphore as a means of synchronization, create it with an initial state of empty. A task blocks by taking a semaphore at a synchronization point, and it remains blocked until the semaphore is given by another task or interrupt service routine.

Synchronization with interrupt service routines is a particularly common need. Binary semaphores can be given, but not taken, from interrupt level. Thus, a task can block at a synchronization point with **semTake()**, and an interrupt service routine can unblock that task with **semGive()**.

In the following example, when **init()** is called, the binary semaphore is created, an interrupt service routine is attached to an event, and a task is spawned to process the event. Task 1 will run until it calls **semTake()**, at which point it will block until an event causes the interrupt service routine to call **semGive()**. When the interrupt service routine completes, task 1 can execute to process the event.

```
SEM_ID semSync;    /* ID of sync semaphore */
init ()
{
    intConnect (... , eventInterruptSvcRout, ...);
    semSync = semBCreate (SEM_Q_FIFO, SEM_EMPTY);
    taskSpawn (... , task1);
}
task1 ()
{
    ...
    semTake (semSync, WAIT_FOREVER);    /* wait for event */
    ...    /* process event */
}
eventInterruptSvcRout ()
{
    ...
    semGive (semSync);    /* let task 1 process event */
    ...
}
```

A **semFlush()** on a binary semaphore will atomically unblock all pended tasks in the semaphore queue, *i.e.*, all tasks will be unblocked at once, before any actually execute.

## CAVEATS

There is no mechanism to give back or reclaim semaphores automatically when tasks are suspended or deleted. Such a mechanism, though desirable, is not currently feasible.

**semCLib**

Without explicit knowledge of the state of the guarded resource or region, reckless automatic reclamation of a semaphore could leave the resource in a partial state. Thus, if a task ceases execution unexpectedly, as with a bus error, currently owned semaphores will not be given back, effectively leaving a resource permanently unavailable. The mutual-exclusion semaphores provided by **semMLib** offer protection from unexpected task deletion.

**INCLUDE FILES**    **semLib.h**

**SEE ALSO**        **semLib**, **semCLib**, **semMLib**, *VxWorks Programmer's Guide: Basic OS*

---

## semCLib

**NAME**            **semCLib** – counting semaphore library

**ROUTINES**        **semCCreate()** - create and initialize a counting semaphore

**DESCRIPTION**    This library provides the interface to VxWorks counting semaphores. Counting semaphores are useful for guarding multiple instances of a resource.

A counting semaphore may be viewed as a cell in memory whose contents keep track of a count. When a task takes a counting semaphore, using **semTake()**, subsequent action depends on the state of the count:

- (1) If the count is non-zero, it is decremented and the calling task continues executing.
- (2) If the count is zero, the task will be blocked, pending the availability of the semaphore. If a timeout is specified and the timeout expires, the pended task will be removed from the queue of pended tasks and enter the ready state with an **ERROR** status. A pended task is ineligible for CPU allocation. Any number of tasks may be pended simultaneously on the same counting semaphore.

When a task gives a semaphore, using **semGive()**, the next available task in the pend queue is unblocked. If no task is pending on this semaphore, the semaphore count is incremented. Note that if a semaphore is given, and a task is unblocked that is of higher priority than the task that called **semGive()**, the unblocked task will preempt the calling task.

A **semFlush()** on a counting semaphore will atomically unblock all pended tasks in the semaphore queue. So all tasks will be made ready before any task actually executes. The count of the semaphore will remain unchanged.

**INTERRUPT USAGE**

Counting semaphores may be given but not taken from interrupt level.

<b>CAVEATS</b>	There is no mechanism to give back or reclaim semaphores automatically when tasks are suspended or deleted. Such a mechanism, though desirable, is not currently feasible. Without explicit knowledge of the state of the guarded resource or region, reckless automatic reclamation of a semaphore could leave the resource in a partial state. Thus, if a task ceases execution unexpectedly, as with a bus error, currently owned semaphores will not be given back, effectively leaving a resource permanently unavailable. The mutual-exclusion semaphores provided by <b>semMLib</b> offer protection from unexpected task deletion.
<b>INCLUDE FILES</b>	<b>semLib.h</b>
<b>SEE ALSO</b>	<b>semLib</b> , <b>semBLib</b> , <b>semMLib</b> , <i>VxWorks Programmer's Guide: Basic OS</i>

---

## semEvLib

<b>NAME</b>	<b>semEvLib</b> – VxWorks events support for semaphores
<b>ROUTINES</b>	<b>semEvStart()</b> - start event notification process for a semaphore <b>semEvStop()</b> - stop event notification process for a semaphore
<b>DESCRIPTION</b>	<p>This library is an extension to <b>eventLib</b>, the events library. Its purpose is to support events for semaphores.</p> <p>The functions in this library are used to control registration of tasks on a semaphore. The routine <b>semEvStart()</b> registers a task and starts the notification process. The function <b>semEvStop()</b> un-registers the task, which stops the notification mechanism.</p> <p>When a task is registered and the semaphore becomes available, the events specified are sent to that task. However, if a <b>semTake()</b> is to be done afterwards, there is no guarantee that the semaphore will still be available.</p>
<b>INCLUDE FILES</b>	<b>semEvLib.h</b>
<b>SEE ALSO</b>	<b>eventLib</b> , <b>semLib</b> , <i>VxWorks Programmer's Guide: Basic OS</i>

---

## semLib

**NAME** **semLib** – general semaphore library

**ROUTINES** **semGive()** - give a semaphore  
**semTake()** - take a semaphore  
**semFlush()** - unblock every task pended on a semaphore  
**semDelete()** - delete a semaphore

**DESCRIPTION** Semaphores are the basis for synchronization and mutual exclusion in VxWorks. They are powerful in their simplicity and form the foundation for numerous VxWorks facilities.

Different semaphore types serve different needs, and while the behavior of the types differs, their basic interface is the same. This library provides semaphore routines common to all VxWorks semaphore types. For all types, the two basic operations are **semTake()** and **semGive()**, the acquisition or relinquishing of a semaphore.

Semaphore creation and initialization is handled by other libraries, depending on the type of semaphore used. These libraries contain full functional descriptions of the semaphore types:

- semBLib** - binary semaphores
- semCLib** - counting semaphores
- semMLib** - mutual exclusion semaphores
- semSmLib** - shared memory semaphores

Binary semaphores offer the greatest speed and the broadest applicability.

The **semLib** library provides all other semaphore operations, including routines for semaphore control, deletion, and information. Semaphores must be validated before any semaphore operation can be undertaken. An invalid semaphore ID results in **ERROR**, and an appropriate **errno** is set.

### SEMAPHORE CONTROL

The **semTake()** call acquires a specified semaphore, blocking the calling task or making the semaphore unavailable. All semaphore types support a timeout on the **semTake()** operation. The timeout is specified as the number of ticks to remain blocked on the semaphore. Timeouts of **WAIT\_FOREVER** and **NO\_WAIT** codify common timeouts. If a **semTake()** times out, it returns **ERROR**. Refer to the library of the specific semaphore type for the exact behavior of this operation.

The **semGive()** call relinquishes a specified semaphore, unblocking a pended task or making the semaphore available. Refer to the library of the specific semaphore type for the exact behavior of this operation.

The **semFlush()** call may be used to atomically unblock all tasks pended on a semaphore queue, *i.e.*, all tasks will be unblocked before any are allowed to run. It may be thought of

as a broadcast operation in synchronization applications. The state of the semaphore is unchanged by the use of **semFlush()**; it is not analogous to **semGive()**.

**SEMAPHORE DELETION**

The **semDelete()** call terminates a semaphore and deallocates any associated memory. The deletion of a semaphore unblocks tasks pending on that semaphore; the routines which were pending return **ERROR**. Take care when deleting semaphores, particularly those used for mutual exclusion, to avoid deleting a semaphore out from under a task that already has taken (owns) that semaphore. Applications should adopt the protocol of only deleting semaphores that the deleting task has successfully taken.

**SEMAPHORE INFORMATION**

The **semInfo()** call is a useful debugging aid, reporting all tasks blocked on a specified semaphore. It provides a snapshot of the queue at the time of the call, but because semaphores are dynamic, the information may be out of date by the time it is available. As with the current state of the semaphore, use of the queue of pending tasks should be restricted to debugging uses only.

**VXWORKS EVENTS** If a task has registered for receiving events with a semaphore, events will be sent when that semaphore becomes available. By becoming available, it is implied that there is a change of state. For a binary semaphore, there is only a change of state when a **semGive()** is done on a semaphore that was taken. For a counting semaphore, there is always a change of state when the semaphore is available, since the count is incremented each time. For a mutex, a **semGive()** can only be performed if the current task is the owner, implying that the semaphore has been taken; thus, there is always a change of state.

**INCLUDE FILES** **semLib.h**

**SEE ALSO** **taskLib, semBLib, semCLib, semMLib, semSmLib, semEvLib, eventLib, VxWorks Programmer's Guide: Basic OS**

---

## semMLib

<b>NAME</b>	<b>semMLib</b> – mutual-exclusion semaphore library
<b>ROUTINES</b>	<b>semMCreate()</b> - create and initialize a mutual-exclusion semaphore <b>semMGiveForce()</b> - give a mutual-exclusion semaphore without restrictions
<b>DESCRIPTION</b>	This library provides the interface to VxWorks mutual-exclusion semaphores. Mutual-exclusion semaphores offer convenient options suited for situations requiring mutually exclusive access to resources. Typical applications include sharing devices and protecting data structures. Mutual-exclusion semaphores are used by many higher-level VxWorks facilities.

The mutual-exclusion semaphore is a specialized version of the binary semaphore, designed to address issues inherent in mutual exclusion, such as recursive access to resources, priority inversion, and deletion safety. The fundamental behavior of the mutual-exclusion semaphore is identical to the binary semaphore (see the manual entry for **semBLib**), except for the following restrictions:

- It can only be used for mutual exclusion.
- It can only be given by the task that took it.
- It may not be taken or given from interrupt level.
- The **semFlush()** operation is illegal.

These last two operations have no meaning in mutual-exclusion situations.

### RECURSIVE RESOURCE ACCESS

A special feature of the mutual-exclusion semaphore is that it may be taken “recursively,” *i.e.*, it can be taken more than once by the task that owns it before finally being released. Recursion is useful for a set of routines that need mutually exclusive access to a resource, but may need to call each other.

Recursion is possible because the system keeps track of which task currently owns a mutual-exclusion semaphore. Before being released, a mutual-exclusion semaphore taken recursively must be given the same number of times it has been taken; this is tracked by means of a count which is incremented with each **semTake()** and decremented with each **semGive()**.

The example below illustrates recursive use of a mutual-exclusion semaphore. Function A requires access to a resource which it acquires by taking **semM**; function A may also need to call function B, which also requires **semM**:

```
SEM_ID semM;  
semM = semMCreate (...);  
funcA ()  
{  
    semTake (semM, WAIT_FOREVER);
```

```
...
funcB ();
...
semGive (semM);
}
funcB ()
{
semTake (semM, WAIT_FOREVER);
...
semGive (semM);
}
```

#### PRIORITY-INVERSION SAFETY

If the option `SEM_INVERSION_SAFE` is selected, the library adopts a priority-inheritance protocol to resolve potential occurrences of “priority inversion,” a problem stemming from the use of semaphores for mutual exclusion. Priority inversion arises when a higher-priority task is forced to wait an indefinite period of time for the completion of a lower-priority task.

Consider the following scenario: T1, T2, and T3 are tasks of high, medium, and low priority, respectively. T3 has acquired some resource by taking its associated semaphore. When T1 preempts T3 and contends for the resource by taking the same semaphore, it becomes blocked. If we could be assured that T1 would be blocked no longer than the time it normally takes T3 to finish with the resource, the situation would not be problematic. However, the low-priority task is vulnerable to preemption by medium-priority tasks; a preempting task, T2, could inhibit T3 from relinquishing the resource. This condition could persist, blocking T1 for an indefinite period of time.

The priority-inheritance protocol solves the problem of priority inversion by elevating the priority of T3 to the priority of T1 during the time T1 is blocked on T3. This protects T3, and indirectly T1, from preemption by T2. Stated more generally, the priority-inheritance protocol assures that a task which owns a resource will execute at the priority of the highest priority task blocked on that resource. Once the task priority has been elevated, it remains at the higher level until all mutual-exclusion semaphores that the task owns are released; then the task returns to its normal, or standard, priority. Hence, the “inheriting” task is protected from preemption by any intermediate-priority tasks.

The priority-inheritance protocol also takes into consideration a task’s ownership of more than one mutual-exclusion semaphore at a time. Such a task will execute at the priority of the highest priority task blocked on any of its owned resources. The task will return to its normal priority only after relinquishing all of its mutual-exclusion semaphores that have the inversion-safety option enabled.

#### SEMAPHORE DELETION

The `semDelete()` call terminates a semaphore and deallocates any associated memory. The deletion of a semaphore unblocks tasks pending on that semaphore; the routines which were pending return `ERROR`. Take special care when deleting mutual-exclusion

semaphores to avoid deleting a semaphore out from under a task that already owns (has taken) that semaphore. Applications should adopt the protocol of only deleting semaphores that the deleting task owns.

#### **TASK-DELETION SAFETY**

If the option **SEM\_DELETE\_SAFE** is selected, the task owning the semaphore will be protected from deletion as long as it owns the semaphore. This solves another problem endemic to mutual exclusion. Deleting a task executing in a critical region can be catastrophic. The resource could be left in a corrupted state and the semaphore guarding the resource would be unavailable, effectively shutting off all access to the resource.

As discussed in **taskLib**, the primitives **taskSafe()** and **taskUnsafe()** offer one solution, but as this type of protection goes hand in hand with mutual exclusion, the mutual-exclusion semaphore provides the option **SEM\_DELETE\_SAFE**, which enables an implicit **taskSafe()** with each **semTake()**, and a **taskUnsafe()** with each **semGive()**. This convenience is also more efficient, as the resulting code requires fewer entrances to the kernel.

#### **CAVEATS**

There is no mechanism to give back or reclaim semaphores automatically when tasks are suspended or deleted. Such a mechanism, though desirable, is not currently feasible. Without explicit knowledge of the state of the guarded resource or region, reckless automatic reclamation of a semaphore could leave the resource in a partial state. Thus if a task ceases execution unexpectedly, as with a bus error, currently owned semaphores will not be given back, effectively leaving a resource permanently unavailable. The **SEM\_DELETE\_SAFE** option partially protects an application, to the extent that unexpected deletions will be deferred until the resource is released.

Because the priority of a task which has been elevated by the taking of a mutual-exclusion semaphore remains at the higher priority until all mutexes held by that task are released, unbounded priority inversion situations can result when nested mutexes are involved. If nested mutexes are required, consider the following alternatives:

1. Avoid overlapping critical regions.
2. Adjust priorities of tasks so that there are no tasks at intermediate priority levels.
3. Adjust priorities of tasks so that priority inheritance protocol is not needed.
4. Manually implement a static priority ceiling protocol using a non-inversion-save mutex. This involves setting all blockers on a mutex to the ceiling priority, then taking the mutex. After **semGive()**, set the priorities back to the base priority. Note that this implementation reduces the queue to a fifo queue.

**INCLUDE FILES**     **semLib.h**

**SEE ALSO**         **semLib**, **semBLib**, **semCLib**, *VxWorks Programmer's Guide: Basic OS*

---

## semOLib

<b>NAME</b>	<b>semOLib</b> – release 4.x binary semaphore library
<b>ROUTINES</b>	<b>semCreate()</b> - create and initialize a release 4.x binary semaphore <b>semInit()</b> - initialize a static binary semaphore <b>semClear()</b> - take a release 4.x semaphore, if the semaphore is available
<b>DESCRIPTION</b>	<p>This library is provided for backward compatibility with VxWorks 4.x semaphores. The semaphores are identical to 5.0 binary semaphores, except that timeouts -- missing or specified -- are ignored.</p> <p>For backward compatibility, <b>semCreate()</b> operates as before, allocating and initializing a 4.x-style semaphore. Likewise, <b>semClear()</b> has been implemented as a <b>semTake()</b>, with a timeout of <b>NO_WAIT</b>.</p> <p>For more information on of the behavior of binary semaphores, see the manual entry for <b>semBLib</b>.</p>
<b>INCLUDE FILES</b>	<b>semLib.h</b>
<b>SEE ALSO</b>	<b>semLib</b> , <b>semBLib</b> , <i>VxWorks Programmer's Guide: Basic OS</i>

---

## semPxBLib

<b>NAME</b>	<b>semPxBLib</b> – semaphore synchronization library (POSIX)
<b>ROUTINES</b>	<b>semPxBLibInit()</b> - initialize POSIX semaphore support <b>sem_init()</b> - initialize an unnamed semaphore (POSIX) <b>sem_destroy()</b> - destroy an unnamed semaphore (POSIX) <b>sem_open()</b> - initialize/open a named semaphore (POSIX) <b>sem_close()</b> - close a named semaphore (POSIX) <b>sem_unlink()</b> - remove a named semaphore (POSIX) <b>sem_wait()</b> - lock (take) a semaphore, blocking if not available (POSIX) <b>sem_trywait()</b> - lock (take) a semaphore, returning error if unavailable (POSIX) <b>sem_post()</b> - unlock (give) a semaphore (POSIX) <b>sem_getvalue()</b> - get the value of a semaphore (POSIX)
<b>DESCRIPTION</b>	This library implements the POSIX 1003.1b semaphore interface. For alternative semaphore routines designed expressly for VxWorks, see the manual page for <b>semLib</b>

**semPxBLib**

and other semaphore libraries mentioned there. POSIX semaphores are counting semaphores; as such they are most similar to the **semCLib** VxWorks-specific semaphores.

The main advantage of POSIX semaphores is portability (to the extent that alternative operating systems also provide these POSIX interfaces). However, VxWorks-specific semaphores provide the following features absent from the semaphores implemented in this library: priority inheritance, task-deletion safety, the ability for a single task to take a semaphore multiple times, ownership of mutual-exclusion semaphores, semaphore timeout, and the choice of queuing mechanism.

POSIX defines both named and unnamed semaphores; **semPxBLib** includes separate routines for creating and deleting each kind. For other operations, applications use the same routines for both kinds of semaphore.

**TERMINOLOGY** The POSIX standard uses the terms *wait* or *lock* where *take* is normally used in VxWorks, and the terms *post* or *unlock* where *give* is normally used in VxWorks. VxWorks documentation that is specific to the POSIX interfaces (such as the remainder of this manual entry, and the manual entries for subroutines in this library) uses the POSIX terminology, in order to make it easier to read in conjunction with other references on POSIX.

**SEMAPHORE DELETION**

The **sem\_destroy()** call terminates an unnamed semaphore and deallocates any associated memory; the combination of **sem\_close()** and **sem\_unlink()** has the same effect for named semaphores. Take care when deleting semaphores, particularly those used for mutual exclusion, to avoid deleting a semaphore out from under a task that has already locked that semaphore. Applications should adopt the protocol of only deleting semaphores that the deleting task has successfully locked. (Similarly, for named semaphores, applications should take care to only close semaphores that the closing task has opened.)

If there are tasks blocked waiting for the semaphore, **sem\_destroy()** fails and sets **errno** to **EBUSY**.

**INCLUDE FILES** **semaphore.h**

**SEE ALSO** POSIX 1003.1b document, **semLib**, *VxWorks Programmer's Guide: Basic OS*

---

## semPxShow

<b>NAME</b>	<b>semPxShow</b> – POSIX semaphore show library
<b>ROUTINES</b>	<b>semPxShowInit()</b> - initialize the POSIX semaphore show facility
<b>DESCRIPTION</b>	This library provides a show routine for POSIX semaphore objects.

---

## semShow

<b>NAME</b>	<b>semShow</b> – semaphore show routines
<b>ROUTINES</b>	<b>semShowInit()</b> - initialize the semaphore show facility <b>semInfo()</b> - get a list of task IDs that are blocked on a semaphore <b>semShow()</b> - show information about a semaphore
<b>DESCRIPTION</b>	<p>This library provides routines to show semaphore statistics, such as semaphore type, semaphore queuing method, tasks pended, <i>etc.</i></p> <p>The routine <b>semShowInit()</b> links the semaphore show facility into the VxWorks system. It is called automatically when the semaphore show facility is configured into VxWorks using either of the following methods:</p> <p>If you use the configuration header files, define <b>INCLUDE_SHOW_ROUTINES</b> in <b>config.h</b>.</p> <p>If you use the Tornado project facility, select <b>INCLUDE_SEM_SHOW</b>.</p>
<b>INCLUDE FILES</b>	<b>semLib.h</b>
<b>SEE ALSO</b>	<b>semLib</b> , <i>VxWorks Programmer's Guide: Basic OS</i>

## semSmLib

<b>NAME</b>	<b>semSmLib</b> – shared memory semaphore library (VxMP Opt.)
<b>ROUTINES</b>	<b>semBSmCreate()</b> - create and initialize shared memory binary semaphore (VxMP Opt.) <b>semCSmCreate()</b> - create and initialize a shared memory counting semaphore (VxMP Opt.)
<b>DESCRIPTION</b>	<p>This library provides the interface to VxWorks shared memory binary and counting semaphores. Once a shared memory semaphore is created, the generic semaphore-handling routines provided in <b>semLib</b> are used to manipulate it. Shared memory binary semaphores are created using <b>semBSmCreate()</b>. Shared memory counting semaphores are created using <b>semCSmCreate()</b>.</p> <p>Shared memory binary semaphores are used to: (1) control mutually exclusive access to multiprocessor-shared data structures, or (2) synchronize multiple tasks running in a multiprocessor system. For general information about binary semaphores, see the manual entry <b>semBLib</b>.</p> <p>Shared memory counting semaphores are used for guarding multiple instances of a resource used by multiple CPUs. For general information about shared counting semaphores, see the manual entry for <b>semCLib</b>.</p> <p>For information about the generic semaphore-handling routines, see the manual entry for <b>semLib</b>.</p>
<b>MEMORY REQUIREMENTS</b>	<p>The semaphore structure is allocated from a dedicated shared memory partition.</p> <p>The shared semaphore dedicated shared memory partition is initialized by the shared memory objects master CPU. The size of this partition is defined by the maximum number of shared semaphores, set in the configuration parameter <b>SM_OBJ_MAX_SEM</b>.</p> <p>This memory partition is common to shared binary and counting semaphores, thus <b>SM_OBJ_MAX_SEM</b> must be set to the sum total of binary and counting semaphores to be used in the system.</p>
<b>RESTRICTIONS</b>	<p>Shared memory semaphores differ from local semaphores in the following ways:</p> <p><b>Interrupt Use:</b> Shared semaphores may not be given, taken, or flushed at interrupt level.</p> <p><b>Deletion:</b> There is no way to delete a shared semaphore and free its associated shared memory. Attempts to delete a shared semaphore return <b>ERROR</b> and set <b>errno</b> to <b>S_smObjLib_NO_OBJECT_DESTROY</b>.</p>

**Queuing Style:**

The shared semaphore queuing style specified when the semaphore is created must be FIFO.

**INTERRUPT LATENCY**

Internally, interrupts are locked while manipulating shared semaphore data structures, thus increasing local CPU interrupt latency.

**CONFIGURATION**

Before routines in this library can be called, the shared memory object facility must be initialized by calling **usrSmObjInit()**. This is done automatically during VxWorks initialization when the component **INCLUDE\_SM\_OBJ** is included.

**AVAILABILITY**

This module is distributed as a component of the unbundled shared memory support option, VxMP.

**INCLUDE FILES**

**semSmLib.h**

**SEE ALSO**

**semLib**, **semBLib**, **semCLib**, **smObjLib**, **semShow**, **usrSmObjInit()**, *VxWorks Programmer's Guide: Shared Memory Objects*, *VxWorks Programmer's Guide: Basic OS*

---

## shellLib

**NAME**

**shellLib** – shell execution routines

**ROUTINES**

**shellInit()** - start the shell  
**shell()** - the shell entry point  
**shellScriptAbort()** - signal the shell to stop processing a script  
**shellHistory()** - display or set the size of shell history  
**shellPromptSet()** - change the shell prompt  
**shellOrigStdSet()** - set the shell's default input/output/error file descriptors  
**shellLock()** - lock access to the shell

**DESCRIPTION**

This library contains the execution support routines for the VxWorks shell. It provides the basic programmer's interface to VxWorks. It is a C-expression interpreter, containing no built-in commands.

The nature, use, and syntax of the shell are more fully described in the "Target Shell" chapter of the *VxWorks Programmer's Guide*.

**INCLUDE FILES**

**shellLib.h**

**SEE ALSO**

**ledLib**, *VxWorks Programmer's Guide: Target Shell*

---

## sigLib

**NAME** sigLib – software signal facility library

**ROUTINES**

- sigInit()** - initialize the signal facilities
- sigqueueInit()** - initialize the queued signal facilities
- sigemptyset()** - initialize a signal set with no signals included (POSIX)
- sigfillset()** - initialize a signal set with all signals included (POSIX)
- sigaddset()** - add a signal to a signal set (POSIX)
- sigdelset()** - delete a signal from a signal set (POSIX)
- sigismember()** - test to see if a signal is in a signal set (POSIX)
- signal()** - specify the handler associated with a signal
- sigaction()** - examine and/or specify the action associated with a signal (POSIX)
- sigprocmask()** - examine and/or change the signal mask (POSIX)
- sigpending()** - retrieve the set of pending signals blocked from delivery (POSIX)
- sigsuspend()** - suspend the task until delivery of a signal (POSIX)
- pause()** - suspend the task until delivery of a signal (POSIX)
- sigtimedwait()** - wait for a signal
- sigwaitinfo()** - wait for real-time signals
- sigwait()** - wait for a signal to be delivered (POSIX)
- sigvec()** - install a signal handler
- sigsetmask()** - set the signal mask
- sigblock()** - add to a set of blocked signals
- raise()** - send a signal to the caller's task
- kill()** - send a signal to a task (POSIX)
- sigqueue()** - send a queued signal to a task

**DESCRIPTION** This library provides a signal interface for tasks. Signals are used to alter the flow control of tasks by communicating asynchronous events within or between task contexts. Any task or interrupt service can “raise” (or send) a signal to a particular task. The task being signaled will immediately suspend its current thread of execution and invoke a task-specified “signal handler” routine. The signal handler is a user-supplied routine that is bound to a specific signal and performs whatever actions are necessary whenever the signal is received. Signals are most appropriate for error and exception handling, rather than as a general purpose intertask communication mechanism.

This library has both a BSD 4.3 and POSIX signal interface. The POSIX interface provides a standardized interface which is more functional than the traditional BSD 4.3 interface. The chart below shows the correlation between BSD 4.3 and POSIX 1003.1 functions. An application should use only one form of interface and not intermix them.

<b>BSD 4.3</b>	<b>POSIX 1003.1</b>
<b>sigmask()</b>	<b>sigemptyset(), sigfillset(), sigaddset(), sigdelset(), sigismember()</b>

BSD 4.3	POSIX 1003.1
<b>sigblock()</b>	<b>sigprocmask()</b>
<b>sigsetmask()</b>	<b>sigprocmask()</b>
<b>pause()</b>	<b>sigsuspend()</b>
<b>sigvec()</b>	<b>sigaction()</b>
(none)	<b>sigpending()</b>
<b>signal()</b>	<b>signal()</b>
<b>kill()</b>	<b>kill()</b>

POSIX 1003.1b (Real-Time Extensions) also specifies a queued-signal facility that involves four additional routines: **sigqueue()**, **sigwaitinfo()**, and **sigtimedwait()**.

In many ways, signals are analogous to hardware interrupts. The signal facility provides a set of 31 distinct signals. A signal can be raised by calling **kill()**, which is analogous to an interrupt or hardware exception. A signal handler is bound to a particular signal with **sigaction()** in much the same way that an interrupt service routine is connected to an interrupt vector with **intConnect()**. Signals are blocked for the duration of the signal handler, just as interrupts are locked out for the duration of the interrupt service routine. Tasks can block the occurrence of certain signals with **sigprocmask()**, just as the interrupt level can be raised or lowered to block out levels of interrupts. If a signal is blocked when it is raised, its handler routine will be called when the signal becomes unblocked.

Several routines (**sigprocmask()**, **sigpending()**, and **sigsuspend()**) take **sigset\_t** data structures as parameters. These data structures are used to specify signal set masks. Several routines are provided for manipulating these data structures: **sigemptyset()** clears all the bits in a **sigset\_t**, **sigfillset()** sets all the bits in a **sigset\_t**, **sigaddset()** sets the bit in a **sigset\_t** corresponding to a particular signal number, **sigdelset()** resets the bit in a **sigset\_t** corresponding to a particular signal number, and **sigismember()** tests to see if the bit corresponding to a particular signal number is set.

## FUNCTION RESTARTING

If a task is pended (for instance, by waiting for a semaphore to become available) and a signal is sent to the task for which the task has a handler installed, then the handler will run before the semaphore is taken. When the handler is done, the task will go back to being pended (waiting for the semaphore). If there was a timeout used for the pend, then the original value will be used again when the task returns from the signal handler and goes back to being pended.

Signal handlers are typically defined as:

```
void sigHandler
(
    int sig,                /* signal number          */
)
{
    ...
}
```

In VxWorks, the signal handler is passed additional arguments and can be defined as:

```
void sigHandler
(
    int sig,                /* signal number          */
    int code,              /* additional code        */
    struct sigcontext *pSigContext /* context of task before signal */
)
{
    ...
}
```

The parameter *code* is valid only for signals caused by hardware exceptions. In this case, it is used to distinguish signal variants. For example, both numeric overflow and zero divide raise SIGFPE (floating-point exception) but have different values for *code*. (Note that when the above VxWorks extensions are used, the compiler may issue warnings.)

#### SIGNAL HANDLER DEFINITION

Signal handling routines must follow one of two specific formats, so that they may be correctly called by the operating system when a signal occurs.

Traditional signal handlers receive the signal number as the sole input parameter. However, certain signals generated by routines which make up the POSIX Real-Time Extensions (P1003.1b) support the passing of an additional application-specific value to the handler routine. These include signals generated by the **sigqueue()** call, by asynchronous I/O, by POSIX real-time timers, and by POSIX message queues.

If a signal handler routine is to receive these additional parameters, SA\_SIGINFO must be set in the **sa\_flags** field of the **sigaction** structure which is a parameter to the **sigaction()** routine. Such routines must take the following form:

```
void sigHandler (int sigNum, siginfo_t * pInfo, void * pContext);
```

Traditional signal handling routines must not set SA\_SIGINFO in the **sa\_flags** field, and must take the form of:

```
void sigHandler (int sigNum);
```

#### EXCEPTION PROCESSING

Certain signals, defined below, are raised automatically when hardware exceptions are encountered. This mechanism allows user-defined exception handlers to be installed. This is useful for recovering from catastrophic events such as bus or arithmetic errors.

Typically, **setjmp()** is called to define the point in the program where control will be restored, and **longjmp()** is called in the signal handler to restore that context. Note that **longjmp()** restores the state of the task's signal mask. If a user-defined handler is not installed or the installed handler returns for a signal raised by a hardware exception, then the task is suspended and a message is logged to the console.

The following is a list of hardware exceptions caught by VxWorks and delivered to the offending task. The user may include the higher-level header file **sigCodes.h** in order to access the appropriate architecture-specific header file containing the code value.

**Motorola 68K**

<b>Signal</b>	<b>Code</b>	<b>Exception</b>
SIGSEGV	NULL	bus error
SIGBUS	BUS_ADDERR	address error
SIGILL	ILL_ILLINSTR_FAULT	illegal instruction
SIGFPE	FPE_INTDIV_TRAP	zero divide
SIGFPE	FPE_CHKINST_TRAP	chk trap
SIGFPE	FPE_TRAPV_TRAP	trapv trap
SIGILL	ILL_PRIVVIO_FAULT	privilege violation
SIGTRAP	NULL	trace exception
SIGEMT	EMT_EMU1010	line 1010 emulator
SIGEMT	EMT_EMU1111	line 1111 emulator
SIGILL	ILL_ILLINSTR_FAULT	coprocessor protocol violation
SIGFMT	NULL	format error
SIGFPE	FPE_FLTBSUN_TRAP	compare unordered
SIGFPE	FPE_FLTINEX_TRAP	inexact result
SIGFPE	FPE_FLTDIV_TRAP	divide by zero
SIGFPE	FPE_FLTUND_TRAP	underflow
SIGFPE	FPE_FLTOPERR_TRAP	operand error
SIGFPE	FPE_FLTOVF_TRAP	overflow
SIGFPE	FPE_FLTNAN_TRAP	signaling "Not A Number"

**MIPS R3000/R4000**

<b>Signal</b>	<b>Code</b>	<b>Exception</b>
SIGBUS	BUS_TLBMOD	TLB modified
SIGBUS	BUS_TLBL	TLB miss on a load instruction
SIGBUS	BUS_TLBS	TLB miss on a store instruction
SIGBUS	BUS_ADEL	address error (bad alignment) on load instr
SIGBUS	BUS_ADES	address error (bad alignment) on store instr
SIGSEGV	SEGV_IBUS	bus error (instruction)
SIGSEGV	SEGV_DBUS	bus error (data)
SIGTRAP	TRAP_SYSCALL	syscall instruction executed
SIGTRAP	TRAP_BP	break instruction executed
SIGILL	ILL_ILLINSTR_FAULT	reserved instruction
SIGILL	ILL_COPROC_UNUSABLE	coprocessor unusable
SIGFPE	FPE_FPA_UIO, SIGFPE	unimplemented FPA operation

Signal	Code	Exception
SIGFPE	FPE_FLTNAN_TRAP	invalid FPA operation
SIGFPE	FPE_FLTDIV_TRAP	FPA divide by zero
SIGFPE	FPE_FLTOVF_TRAP	FPA overflow exception
SIGFPE	FPE_FLTUND_TRAP	FPA underflow exception
SIGFPE	FPE_FLTINEX_TRAP	FPA inexact operation

Intel i386/i486

Signal	Code	Exception
SIGILL	ILL_DIVIDE_ERROR	divide error
SIGEMT	EMT_DEBUG	debugger call
SIGILL	ILL_NON_MASKABLE	NMI interrupt
SIGEMT	EMT_BREAKPOINT	breakpoint
SIGILL	ILL_OVERFLOW	INTO-detected overflow
SIGILL	ILL_BOUND	bound range exceeded
SIGILL	ILL_INVALID_OPCODE	invalid opcode
SIGFPE	FPE_NO_DEVICE	device not available
SIGILL	ILL_DOUBLE_FAULT	double fault
SIGFPE	FPE_CP_OVERRUN	coprocessor segment overrun
SIGILL	ILL_INVALID_TSS	invalid task state segment
SIGBUS	BUS_NO_SEGMENT	segment not present
SIGBUS	BUS_STACK_FAULT	stack exception
SIGILL	ILL_PROTECTION_FAULT	general protection
SIGBUS	BUS_PAGE_FAULT	page fault
SIGILL	ILL_RESERVED	(intel reserved)
SIGFPE	FPE_CP_ERROR	coprocessor error
SIGBUS	BUS_ALIGNMENT	alignment check

PowerPC

Signal	Code	Exception
SIGBUS	_EXC_OFF_MACH	machine check
SIGBUS	_EXC_OFF_INST	instruction access
SIGBUS	_EXC_OFF_ALIGN	alignment
SIGILL	_EXC_OFF_PROG	program
SIGBUS	_EXC_OFF_DATA	data access
SIGFPE	_EXC_OFF_FPU	floating point unavailable
SIGTRAP	_EXC_OFF_DBG	debug exception (PPC403)
SIGTRAP	_EXC_OFF_INST_BRK	inst. breakpoint (PPC603, PPCEC603, PPC604)
SIGTRAP	_EXC_OFF_TRACE	trace (PPC603, PPCEC603, PPC604, PPC860)
SIGBUS	_EXC_OFF_CRTL	critical interrupt (PPC403)

Signal	Code	Exception
SIGILL	_EXC_OFF_SYSCALL	system call

## Hitachi SH770x

Signal	Code	Exception
SIGSEGV	TLB_LOAD_MISS	TLB miss/invalid (load)
SIGSEGV	TLB_STORE_MISS	TLB miss/invalid (store)
SIGSEGV	TLB_INITIAL_PAGE_WRITE	Initial page write
SIGSEGV	TLB_LOAD_PROTEC_VIOLATION	TLB protection violation (load)
SIGSEGV	TLB_STORE_PROTEC_VIOLATION	TLB protection violation (store)
SIGBUS	BUS_LOAD_ADDRESS_ERROR	Address error (load)
SIGBUS	BUS_STORE_ADDRESS_ERROR	Address error (store)
SIGILL	ILLEGAL_INSTR_GENERAL	general illegal instruction
SIGILL	ILLEGAL_SLOT_INSTR	slot illegal instruction
SIGFPE	FPE_INTDIV_TRAP	integer zero divide

## Hitachi SH7604/SH704x/SH703x/SH702x

Signal	Code	Exception
SIGILL	ILL_ILINSTR_GENERAL	general illegal instruction
SIGILL	ILL_ILINSTR_SLOT	slot illegal instruction
SIGBUS	BUS_ADDERR_CPU	CPU address error
SIGBUS	BUS_ADDERR_DMA	DMA address error
SIGFPE	FPE_INTDIV_TRAP	integer zero divide

Two signals are provided for application use: **SIGUSR1** and **SIGUSR2**. VxWorks will never use these signals; however, other signals may be used by VxWorks in the future.

**INCLUDE FILES**     `signal.h`

**SEE ALSO**            `intLib`, *IEEE POSIX 1003.1b*, *VxWorks Programmer's Guide: Basic OS*

---

## smMemLib

**NAME**                `smMemLib` – shared memory management library (VxMP Opt.)

**ROUTINES**          `memPartSmCreate()` - create a shared memory partition  
`smMemAddToPool()` - add memory to shared memory system partition  
`smMemOptionsSet()` - set debug options for shared memory system partition

**smMemMalloc()** - allocate block of memory from shared memory system partition  
**smMemCalloc()** - allocate memory for array from shared memory system partition  
**smMemRealloc()** - reallocate block of memory from shared memory system partition  
**smMemFree()** - free a shared memory system partition block of memory  
**smMemFindMax()** - find largest free block in shared memory system partition

**DESCRIPTION**

This library provides facilities for managing the allocation of blocks of shared memory from ranges of memory called shared memory partitions. The routine **memPartSmCreate()** is used to create shared memory partitions in the shared memory pool. The created partition can be manipulated using the generic memory partition calls, **memPartAlloc()**, **memPartFree()**, *etc.* (for a complete list of these routines, see the manual entry for **memPartLib**). The maximum number of partitions that can be created is determined by the configuration parameter **SM\_OBJ\_MAX\_MEM\_PART**.

The **smMem...()** routines provide an easy-to-use interface to the shared memory system partition. The shared memory system partition is created when the shared memory object facility is initialized.

Shared memory management information and statistics display routines are provided by **smMemShow**.

The allocation of memory, using **memPartAlloc()** in the general case and **smMemMalloc()** for the shared memory system partition, is done with a first-fit algorithm. Adjacent blocks of memory are coalesced when freed using **memPartFree()** and **smMemFree()**.

There is a 28-byte overhead per allocated block (architecture dependent), and allocated blocks are aligned on a 16-byte boundary.

All memory used by the shared memory facility must be in the same address space, that is, it must be reachable from all the CPUs with the same offset as the one used for the shared memory anchor.

**CONFIGURATION**

Before routines in this library can be called, the shared memory objects facility must be initialized by a call to **usrSmObjInit()**, which is found in **target/config/comps/src/usrSmObj.c**. This is done automatically by VxWorks when the **INCLUDE\_SM\_OBJ** component is included.

**ERROR OPTIONS**

Various debug options can be selected for each partition using **memPartOptionsSet()** and **smMemOptionsSet()**. Two kinds of errors are detected: attempts to allocate more memory than is available, and bad blocks found when memory is freed. In both cases, options can be selected for system actions to take place when the error is detected: (1) return the error status, (2) log an error message and return the error status, or (3) log an error message and suspend the calling task.

One of the following options can be specified to determine the action to be taken when there is an attempt to allocate more memory than is available in the partition:

**MEM\_ALLOC\_ERROR\_RETURN**

just return the error status to the calling task.

**MEM\_ALLOC\_ERROR\_LOG\_MSG**

log an error message and return the status to the calling task.

**MEM\_ALLOC\_ERROR\_LOG\_AND\_SUSPEND**

log an error message and suspend the calling task.

The following option is specified by default to check every block freed to the partition. If this option is specified, **memPartFree()** and **smMemFree()** will make a consistency check of various pointers and values in the header of the block being freed.

**MEM\_BLOCK\_CHECK**

check each block freed.

One of the following options can be specified to determine the action to be taken when a bad block is detected when freed. These options apply only if the **MEM\_BLOCK\_CHECK** option is selected.

**MEM\_BLOCK\_ERROR\_RETURN**

just return the status to the calling task.

**MEM\_BLOCK\_ERROR\_LOG\_MSG**

log an error message and return the status to the calling task.

**MEM\_BLOCK\_ERROR\_LOG\_AND\_SUSPEND**

log an error message and suspend the calling task.

The default options when a shared partition is created are

**MEM\_ALLOC\_ERROR\_LOG\_MSG**, **MEM\_BLOCK\_CHECK**, **MEM\_BLOCK\_ERROR\_RETURN**.

When setting options for a partition with **memPartOptionsSet()** or **smMemOptionsSet()**, use the logical OR operator between each specified option to construct the *options* parameter. For example:

```
memPartOptionsSet (myPartId, MEM_ALLOC_ERROR_LOG_MSG |  
                   MEM_BLOCK_CHECK |  
                   MEM_BLOCK_ERROR_LOG_MSG);
```

<b>AVAILABILITY</b>	This module is distributed as a component of the unbundled shared memory objects support option, VxMP.
<b>INCLUDE FILES</b>	<b>smMemLib.h</b>
<b>SEE ALSO</b>	<b>smMemShow</b> , <b>memLib</b> , <b>memPartLib</b> , <b>smObjLib</b> , <b>usrSmObjInit()</b> , <i>VxWorks Programmer's Guide: Shared Memory Objects</i>

## smMemShow

<b>NAME</b>	<b>smMemShow</b> – shared memory management show routines (VxMP Opt.)
<b>ROUTINES</b>	<b>smMemShow()</b> - show the shared memory system partition blocks and statistics (VxMP Opt.)
<b>DESCRIPTION</b>	This library provides routines to show the statistics on a shared memory system partition. General shared memory management routines are provided by <b>smMemLib</b> .
<b>CONFIGURATION</b>	The routines in this library are included by default if the component <b>INCLUDE_SM_OBJ</b> is included.
<b>AVAILABILITY</b>	This module is distributed as a component of the unbundled shared memory objects support option, VxMP.
<b>INCLUDE FILES</b>	<b>smLib.h</b> , <b>smObjLib.h</b> , <b>smMemLib.h</b>
<b>SEE ALSO</b>	<b>smMemLib</b> , <i>VxWorks Programmer's Guide: Shared Memory Objects</i>

---

## smNameLib

<b>NAME</b>	<b>smNameLib</b> – shared memory objects name database library (VxMP Opt.)
<b>ROUTINES</b>	<b>smNameAdd()</b> - add a name to the shared memory name database (VxMP Opt.) <b>smNameFind()</b> - look up a shared memory object by name (VxMP Opt.) <b>smNameFindByValue()</b> - look up a shared memory object by value (VxMP Opt.) <b>smNameRemove()</b> - remove an object from the shared memory objects name database (VxMP Opt.)
<b>DESCRIPTION</b>	<p>This library provides facilities for managing the shared memory objects name database. The shared memory objects name database associates a name and object type with a value and makes that information available to all CPUs. A name is an arbitrary, null-terminated string. An object type is a small integer, and its value is a global (shared) ID or a global shared memory address.</p> <p>Names are added to the shared memory name database with <b>smNameAdd()</b>. They are removed by <b>smNameRemove()</b>.</p> <p>Objects in the database can be accessed by either name or value. The routine <b>smNameFind()</b> searches the shared memory name database for an object of a specified</p>

name. The routine `smNameFindByValue()` searches the shared memory name database for an object of a specified identifier or address.

Name database contents can be viewed using `smNameShow()`.

The maximum number of names to be entered in the database is defined in the configuration parameter `SM_OBJ_MAX_NAME`. This value is used to determine the size of a dedicated shared memory partition from which name database fields are allocated.

The estimated memory size required for the name database can be calculated as follows:

```
name database pool size = SM_OBJ_MAX_NAME * 40 (bytes)
```

The display facility for the shared memory objects name database is provided by the `smNameShow` module.

#### EXAMPLE

The following code fragment allows a task on one CPU to enter the name, associated ID, and type of a created shared semaphore into the name database. Note that CPU numbers can belong to any CPU using the shared memory objects facility.

On CPU 1:

```
#include "vxWorks.h"
#include "semLib.h"
#include "smNameLib.h"
#include "semSmLib.h"
#include "stdio.h"
testSmSem1 (void)
{
    SEM_ID smSemId;
    /* create a shared semaphore */
    if ((smSemId = semBSmCreate(SEM_Q_FIFO, SEM_EMPTY)) == NULL)
    {
        printf ("Shared semaphore creation error.");
        return (ERROR);
    }
    /*
     * make created semaphore Id available to all CPUs in
     * the system by entering its name in shared name database.
     */
    if (smNameAdd ("smSem", smSemId, T_SM_SEM_B) != OK )
    {
        printf ("Cannot add smSem into shared database.");
        return (ERROR);
    }
    ...
    /* now use the semaphore */
    semGive (smSemId);
    ...
}
```

```
}
```

On CPU 2:

```
#include "vxWorks.h"
#include "semLib.h"
#include "smNameLib.h"
#include "stdio.h"
testSmSem2 (void)
{
    SEM_ID smSemId;
    int    objType; /* place holder for smNameFind() object type */
    /* get semaphore ID from name database */

    smNameFind ("smSem", (void **) &smSemId, &objType, WAIT_FOREVER);
    ...
    /* now that we have the shared semaphore ID, take it */

    semTake (smSemId, WAIT_FOREVER);
    ...
}
```

**CONFIGURATION** Before routines in this library can be called, the shared memory object facility must be initialized by calling **usrSmObjInit()**. This is done automatically during VxWorks initialization when the component **INCLUDE\_SM\_OBJ** is included.

**AVAILABILITY** This module is distributed as a component of the unbundled shared memory objects support option, VxMP.

**INCLUDE FILES** **smNameLib.h**

**SEE ALSO** **smNameShow**, **smObjLib**, **smObjShow**, **usrSmObjInit()**, *VxWorks Programmer's Guide: Shared Memory Objects*

---

## smNameShow

**NAME** **smNameShow** – shared memory objects name database show routines (VxMP Opt.)

**ROUTINES** **smNameShow()** - show the contents of the shared memory objects name database (VxMP Opt.)

<b>DESCRIPTION</b>	This library provides a routine to show the contents of the shared memory objects name database. The shared memory objects name database facility is provided by the <b>smNameLib</b> module.
<b>CONFIGURATION</b>	The routines in this library are included by default if the component <b>INCLUDE_SM_OBJ</b> is included.
<b>AVAILABILITY</b>	This module is distributed as a component of the unbundled shared memory objects support option, VxMP.
<b>INCLUDE FILES</b>	<b>smNameLib.h</b>
<b>SEE ALSO</b>	<b>smNameLib</b> , <b>smObjLib</b> , <i>VxWorks Programmer's Guide: Shared Memory Objects</i>

---

## smNetLib

<b>NAME</b>	<b>smNetLib</b> – VxWorks interface to shared memory network (backplane) driver
<b>ROUTINES</b>	No Callable Routines.
<b>DESCRIPTION</b>	<p>This library implements the VxWorks-specific portions of the shared memory network interface driver. It provides the interface between VxWorks and the network driver modules (<i>e.g.</i>, how the OS initializes and attaches the driver, interrupt handling, <i>etc.</i>), as well as VxWorks-dependent system calls.</p> <p>There are no user-callable routines.</p> <p>The backplane master initializes the backplane shared memory and network structures by first calling <b>smNetInit()</b>. Once the backplane has been initialized, all processors can be attached to the shared memory network via the <b>smNetAttach()</b> routine. Both <b>smNetInit()</b> and <b>smNetAttach()</b> are called automatically during system initialization when backplane parameters are specified in the boot line.</p> <p>For detailed information refer to <i>VxWorks Network Programmer's Guide: Data Link Layer Network Components</i>.</p>
<b>INCLUDE FILES</b>	<b>smNetLib.h</b> , <b>smPktLib.h</b> , <b>smUtilLib.h</b>
<b>SEE ALSO</b>	<b>ifLib</b> , <b>if_sm</b> , <i>VxWorks Network Programmer's Guide</i>

## smNetShow

<b>NAME</b>	<b>smNetShow</b> – shared memory network driver show routines
<b>ROUTINES</b>	<b>smNetShow()</b> - show information about a shared memory network
<b>DESCRIPTION</b>	This library provides show routines for the shared memory network interface driver.  The <b>smNetShow()</b> routine is provided as a diagnostic aid to show current shared memory network status.
<b>INCLUDE FILES</b>	<b>smNetLib.h</b> , <b>smPktLib.h</b>
<b>SEE ALSO</b>	<b>if_sm</b> , <b>smNetLib</b> , <b>smPktLib</b> , <i>VxWorks Network Programmer's Guide</i>

---

## smObjLib

<b>NAME</b>	<b>smObjLib</b> – shared memory objects library (VxMP Opt.)
<b>ROUTINES</b>	<b>smObjLibInit()</b> - install the shared memory objects facility <b>smObjSetup()</b> - initialize the shared memory objects facility <b>smObjInit()</b> - initialize a shared memory objects descriptor <b>smObjAttach()</b> - attach the calling CPU to the shared memory objects facility <b>smObjLocalToGlobal()</b> - convert a local address to a global address <b>smObjGlobalToLocal()</b> - convert a global address to a local address <b>smObjTimeoutLogEnable()</b> - control logging of failed attempts to take a spin-lock
<b>DESCRIPTION</b>	This library contains miscellaneous functions used by the shared memory objects facility (VxMP). Shared memory objects provide high-speed synchronization and communication among tasks running on separate CPUs that have access to a common shared memory. Shared memory objects are system objects ( <i>e.g.</i> , semaphores and message queues) that can be used across processors.  The main uses of shared memory objects are inter-processor synchronization, mutual exclusion on multiprocessor shared data structures, and high-speed data exchange.  Routines for displaying shared memory objects statistics are provided by the <b>smObjShow</b> module.
<b>SHARED MEMORY MASTER CPU</b>	One CPU node acts as the shared memory objects master. This CPU initializes the shared memory area and sets up the shared memory anchor. These steps are performed by the

master calling **smObjSetup()**. This routine should be called only once by the master CPU. Usually **smObjSetup()** is called from **usrSmObjInit()**. (See *Configuration* below.)

Once **smObjSetup()** has completed successfully, there is little functional difference between the master CPU and other CPUs using shared memory objects, except that the master is responsible for maintaining the heartbeat in the shared memory objects header.

#### ATTACHING TO SHARED MEMORY

Each CPU, master or non-master, that will use shared memory objects must attach itself to the shared memory objects facility, which must already be initialized.

Before it can attach to a shared memory region, each CPU must allocate and initialize a shared memory descriptor (**SM\_DESC**), which describes the individual CPU's attachment to the shared memory objects facility. Since the shared memory descriptor is used only by the local CPU, it is not necessary for the descriptor itself to be located in shared memory. In fact, it is preferable for the descriptor to be allocated from the CPU's local memory, since local memory is usually more efficiently accessed.

The shared memory descriptor is initialized by calling **smObjInit()**. This routine takes a number of parameters which specify the characteristics of the calling CPU and its access to shared memory.

Once the shared memory descriptor has been initialized, the CPU can attach itself to the shared memory region. This is done by calling **smObjAttach()**.

When **smObjAttach()** is called, it verifies that the shared memory anchor contains the value **SM\_READY** and that the heartbeat located in the shared memory objects header is incrementing. If either of these conditions is not met, the routine will check periodically until either **SM\_READY** or an incrementing heartbeat is recognized or a time limit is reached. The limit is expressed in seconds, and 600 seconds (10 minutes) is the default. If the time limit is reached before **SM\_READY** or a heartbeat is found, **ERROR** is returned and **errno** is set to **S\_smLib\_DOWN**.

#### ADDRESS CONVERSION

This library also provides routines for converting between local and global shared memory addresses, **smObjLocalToGlobal()** and **smObjGlobalToLocal()**. A local shared memory address is the address required by the local CPU to reach a location in shared memory. A global shared memory address is a value common to all CPUs in the system used to reference a shared memory location. A global shared memory address is always an offset from the shared memory anchor.

#### SPIN-LOCK MECHANISM

The shared memory objects facilities use a spin-lock mechanism based on an indivisible read-modify-write (RMW) operation on a shared memory location which acts as a low-level mutual exclusion device. The spin-lock mechanism is called with a system-wide configuration parameter, **SM\_OBJ\_MAX\_TRIES**, which specifies the maximum number of RMW tries on a spin-lock location.

Care must be taken that the number of RMW tries on a spin-lock on a particular CPU never reaches **SM\_OBJ\_MAX\_TRIES**, otherwise system behavior becomes unpredictable. The default value should be sufficient for reliable operation.

The routine **smObjTimeoutLogEnable()** can be used to enable or disable the printing of a message should a shared memory object call fail while trying to take a spin-lock.

#### **RELATION TO BACKPLANE DRIVER**

Shared memory objects and the shared memory network (backplane) driver use common underlying shared memory utilities. They also use the same anchor, the same shared memory header, and the same interrupt when they are used at the same time.

#### **LIMITATIONS**

A maximum of twenty CPUs can be used concurrently with shared memory objects. Each CPU in the system must have a hardware test-and-set (TAS) mechanism, which is called via the system-dependent routine **sysBusTas()**.

The use of shared memory objects raises interrupt latency, because internal mechanisms lock interrupts while manipulating critical shared data structures. Interrupt latency does not depend on the number of objects or CPUs used.

#### **GETTING STATUS INFORMATION**

The routine **smObjShow()** displays useful information regarding the current status of shared memory objects, including the number of tasks using shared objects, shared semaphores, and shared message queues, the number of names in the database, and also the maximum number of tries to get spin-lock access for the calling CPU.

#### **CONFIGURATION**

When the component **INCLUDE\_SM\_OBJ** is included, the init and setup routines in this library are called automatically during VxWorks initialization.

#### **AVAILABILITY**

This module is distributed as a component of the unbundled shared memory objects support option, VxMP.

#### **INCLUDE FILES**

**smObjLib.h**

#### **SEE ALSO**

**smObjShow**, **semSmLib**, **msgQSmLib**, **smMemLib**, **smNameLib**, **usrSmObjInit()**,  
*VxWorks Programmer's Guide: Shared Memory Objects*

---

## smObjShow

<b>NAME</b>	<b>smObjShow</b> – shared memory objects show routines (VxMP Opt.)
<b>ROUTINES</b>	<b>smObjShow()</b> - display the current status of shared memory objects (VxMP Opt.)
<b>DESCRIPTION</b>	This library provides routines to show shared memory object statistics, such as the current number of shared tasks, semaphores, message queues, <i>etc.</i>
<b>CONFIGURATION</b>	The routines in this library are included by default if the component <b>INCLUDE_SM_OBJ</b> is included.
<b>AVAILABILITY</b>	This module is distributed as a component of the unbundled shared memory objects support option, VxMP.
<b>INCLUDE FILES</b>	<b>smObjLib.h</b>
<b>SEE ALSO</b>	<b>smObjLib</b> , <i>VxWorks Programmer's Guide: Shared Memory Objects</i>

---

## sntpcLib

<b>NAME</b>	<b>sntpcLib</b> – Simple Network Time Protocol (SNTP) client library
<b>ROUTINES</b>	<b>sntpcTimeGet()</b> - retrieve the current time from a remote source
<b>DESCRIPTION</b>	This library implements the client side of the Simple Network Time Protocol (SNTP), a protocol that allows a system to maintain the accuracy of its internal clock based on time values reported by one or more remote sources. The library is included in the VxWorks image if <b>INCLUDE_SNTPC</b> is defined at the time the image is built.
<b>USER INTERFACE</b>	The <b>sntpcTimeGet()</b> routine retrieves the time reported by a remote source and converts that value for POSIX-compliant clocks. The routine will either send a request and extract the time from the reply, or it will wait until a message is received from an SNTP/NTP server executing in broadcast mode.
<b>INCLUDE FILES</b>	<b>sntpcLib.h</b>
<b>SEE ALSO</b>	<b>clockLib</b> , RFC 1769

---

## sntpLib

<b>NAME</b>	<b>sntpLib</b> – Simple Network Time Protocol (SNTP) server library
<b>ROUTINES</b>	<b>sntpClockSet()</b> - assign a routine to access the reference clock <b>sntpNsecToFraction()</b> - convert portions of a second to NTP format <b>sntpConfigSet()</b> - change SNTP server broadcast settings
<b>DESCRIPTION</b>	This library implements the server side of the Simple Network Time Protocol (SNTP), a protocol that allows a system to maintain the accuracy of its internal clock based on time values reported by one or more remote sources. The library is included in the VxWorks image if <code>INCLUDE_SNTPS</code> is defined at the time the image is built.
<b>USER INTERFACE</b>	<p>The routine <b>sntpInit()</b> is called automatically during system startup when the SNTP server library is included in the VxWorks image. Depending on the value of <code>SNTPS_MODE</code>, the server executes in either a passive or an active mode. When <code>SNTPS_MODE</code> is set to <code>SNTP_PASSIVE</code> (0x2), the server waits for requests from clients, and sends replies containing an NTP timestamp. When the mode is set to <code>SNTP_ACTIVE</code> (0x1), the server transmits NTP timestamp information at fixed intervals.</p> <p>When executing in active mode, the SNTP server uses the <code>SNTPS_DSTADDR</code> and <code>SNTPS_INTERVAL</code> definitions to determine the target IP address and broadcast interval. By default, the server transmits the timestamp information to the local subnet broadcast address every 64 seconds. These settings can be changed with <b>sntpConfigSet()</b>. The SNTP server operating in active mode will still respond to client requests.</p> <p>The <code>SNTP_PORT</code> definition in assigns the source and destination UDP port. The default port setting is 123 as specified by the relevant RFC. Finally, the SNTP server requires access to a reliable external time source. The <code>SNTPS_TIME_HOOK</code> constant specifies the name of a routine with the following interface:</p> <pre><b>STATUS</b> sntpTimeHook (int request, void *pBuffer);</pre> <p>This routine can be assigned directly by altering the value of <code>SNTPS_TIME_HOOK</code> or can be installed by a call to <b>sntpClockSet()</b>. The manual pages for <b>sntpClockSet()</b> describe the parameters and required operation of the timestamp retrieval routine. Until this routine is specified, the SNTP server will not provide timestamp information.</p>
<b>VXWORKS AE PROTECTION DOMAINS</b>	Under VxWorks AE, the SNTP server can run in the kernel protection domain only. The <code>SNTPS_TIME_HOOK MUST</code> , if used, must reference a function in the kernel protection domain. This restriction does not apply under non-AE versions of VxWorks.
<b>INCLUDE FILES</b>	<b>sntpLib.h</b>
<b>SEE ALSO</b>	<b>sntpcLib</b> , RFC 1769

---

## sockLib

<b>NAME</b>	<b>sockLib</b> – generic socket library
<b>ROUTINES</b>	<b>socket()</b> - open a socket <b>bind()</b> - bind a name to a socket <b>listen()</b> - enable connections to a socket <b>accept()</b> - accept a connection from a socket <b>connect()</b> - initiate a connection to a socket <b>connectWithTimeout()</b> - attempt socket connection within a specified duration <b>sendto()</b> - send a message to a socket <b>send()</b> - send data to a socket <b>sendmsg()</b> - send a message to a socket <b>recvfrom()</b> - receive a message from a socket <b>recv()</b> - receive data from a socket <b>recvmsg()</b> - receive a message from a socket <b>setsockopt()</b> - set socket options <b>getsockopt()</b> - get socket options <b>getsockname()</b> - get a socket name <b>getpeername()</b> - get the name of a connected peer <b>shutdown()</b> - shut down a network connection
<b>DESCRIPTION</b>	<p>This library provides UNIX BSD 4.4 compatible socket calls. Use these calls to open, close, read, and write sockets. These sockets can join processes on the same CPU or on different CPUs between which there is a network connection. The calling sequences of these routines are identical to their equivalents under UNIX BSD 4.4.</p> <p>However, although the socket interface is compatible with VxWorks, the VxWorks environment does affect how you use sockets. Specifically, the globally accessible file descriptors available in the single address space world of VxWorks require that you take extra precautions when closing a file descriptor.</p> <p>You must make sure that you do not close the file descriptor on which a task is pending during an <b>accept()</b>. Although the <b>accept()</b> on the closed file descriptor sometimes returns with an error, the <b>accept()</b> can also fail to return at all. Thus, if you need to be able to close a socket connections file descriptor asynchronously, you may need to set up a semaphore-based locking mechanism that prevents the close while an <b>accept()</b> is pending on the file descriptor.</p>
<b>ADDRESS FAMILY</b>	VxWorks sockets support only the Internet Domain address family. Use <b>AF_INET</b> for the <i>domain</i> argument in subroutines that require it. There is no support for the UNIX Domain address family.
<b>IOCTL FUNCTIONS</b>	Sockets respond to the following <b>ioctl()</b> functions. These functions are defined in the header files <b>ioLib.h</b> and <b>ioctl.h</b> .

#### FIONBIO

Turns on/off non-blocking I/O.

```
on = TRUE;  
status = ioctl (sFd, FIONBIO, &on);
```

#### FIONREAD

Reports the number of read-ready bytes available on the socket. On the return of `ioctl()`, *bytesAvailable* has the number of bytes available to read from the socket.

```
status = ioctl (sFd, FIONREAD, &bytesAvailable);
```

#### SIOCATMARK

Reports whether there is out-of-band data to be read from the socket. On the return of `ioctl()`, *atMark* is `TRUE` (1) if there is out-of-band data. Otherwise, it is `FALSE` (0).

```
status = ioctl (sFd, SIOCATMARK, &atMark);
```

To use this feature, include the following component: `INCLUDE_BSD_SOCKET`.

**INCLUDE FILES**     `types.h, mbuf.h, socket.h, socketvar.h`

**SEE ALSO**         `netLib`, *UNIX Network Programming*, by W. Richard Stevens

---

## spyLib

**NAME**             `spyLib` – spy CPU activity library

**ROUTINES**        `spyLibInit()` - initialize task CPU utilization tool package

**DESCRIPTION**    This library provides a facility to monitor tasks' use of the CPU. The primary interface routine, `spy()`, periodically calls `spyReport()` to display the amount of CPU time utilized by each task, the amount of time spent at interrupt level, the amount of time spent in the kernel, and the amount of idle time. It also displays the total usage since the start of `spy()` (or the last call to `spyClkStart()`), and the change in usage since the last `spyReport()`.

CPU usage can also be monitored manually by calling `spyClkStart()` and `spyReport()`, instead of `spy()`. In this case, `spyReport()` provides a one-time report of the same information provided by `spy()`.

Data is gathered by an interrupt-level routine that is connected by `spyClkStart()` to the auxiliary clock. Currently, this facility cannot be used with CPUs that have no auxiliary clock. Interrupts that are at a higher level than the auxiliary clock's interrupt level cannot be monitored.

All user interface routine except `spyLibInit()` are available through `usrLib`.

## EXAMPLE

The following call:

```
-> spy 10, 200
```

will generate a report in the following format every 10 seconds, gathering data at the rate of 200 times per second.

NAME	ENTRY	TID	PRI	total % (ticks)	delta % (ticks)
tExcTask	_excTask	fb58	0	0% ( 0)	0% ( 0)
tLogTask	_logTask	fa6e0	0	0% ( 0)	0% ( 0)
tShell	_shell	e28a8	1	0% ( 4)	0% ( 0)
tRlogind	_rlogind	f08dc	2	0% ( 0)	0% ( 0)
tRlogOutTask	_rlogOutTa	e93e0	2	2% ( 173)	2% ( 46)
tRlogInTask	_rlogInTas	e7f10	2	0% ( 0)	0% ( 0)
tSpyTask	_spyTask	ffe9c	5	1% ( 116)	1% ( 28)
tNetTask	_netTask	f3e2c	50	0% ( 4)	0% ( 1)
tPortmapd	_portmapd	ef240	100	0% ( 0)	0% ( 0)
KERNEL				1% ( 105)	0% ( 10)
INTERRUPT				0% ( 0)	0% ( 0)
IDLE				95% ( 7990)	95% ( 1998)
TOTAL				99% ( 8337)	98% ( 2083)

The “total” column reflects CPU activity since the initial call to `spy()` or the last call to `spyClkStart()`. The “delta” column reflects activity since the previous report. A call to `spyReport()` will produce a single report; however, the initial auxiliary clock interrupts and data collection must first be started using `spyClkStart()`.

Data collection/clock interrupts and periodic reporting are stopped by calling:

```
-> spyStop
```

INCLUDE FILES `spyLib.h`

SEE ALSO `usrLib`

---

## symLib

NAME `symLib` – symbol table subroutine library

ROUTINES

- `symLibInit()` - initialize the symbol table library
- `symTblCreate()` - create a symbol table
- `symTblDelete()` - delete a symbol table
- `symAdd()` - create and add a symbol to a symbol table, including a group number
- `symRemove()` - remove a symbol from a symbol table

**symLib**

**symFindByName()** - look up a symbol by name  
**symFindByNameAndType()** - look up a symbol by name and type  
**symByValueFind()** - look up a symbol by value  
**symByValueAndTypeFind()** - look up a symbol by value and type  
**symFindByValue()** - look up a symbol by value  
**symFindByValueAndType()** - look up a symbol by value and type  
**symEach()** - call a routine to examine each entry in a symbol table

**DESCRIPTION**

This library provides facilities for managing symbol tables. A symbol table associates a name and type with a value. A name is simply an arbitrary, null-terminated string. A symbol type is a small integer (typedef **SYM\_TYPE**), and its value is a pointer. Though commonly used as the basis for object loaders, symbol tables may be used whenever efficient association of a value with a name is needed.

If you use the **symLib** subroutines to manage symbol tables local to your own applications, the values for **SYM\_TYPE** objects are completely arbitrary; you can use whatever one-byte integers are appropriate for your application.

If you use the **symLib** subroutines to manipulate the VxWorks system symbol table (whose ID is recorded in the global **sysSymTbl**), the values for **SYM\_TYPE** are **SYM\_UNDF**, **SYM\_LOCAL**, **SYM\_GLOBAL**, **SYM\_ABS**, **SYM\_TEXT**, **SYM\_DATA**, **SYM\_BSS**, and **SYM\_COMM** (defined in **symbol.h**).

Tables are created with **symTblCreate()**, which returns a symbol table ID. This ID serves as a handle for symbol table operations, including the adding to, removing from, and searching of tables. All operations on a symbol table are interlocked by means of a mutual-exclusion semaphore in the symbol table structure. Tables are deleted with **symTblDelete()**.

Symbols are added to a symbol table with **symAdd()**. Each symbol in the symbol table has a name, a value, and a type. Symbols are removed from a symbol table with **symRemove()**.

Symbols can be accessed by either name or value. The routine **symFindByName()** searches the symbol table for a symbol with a specified name. The routine **symByValueFind()** finds a symbol with a specified value or, if there is no symbol with the same value, the symbol in the table with the next lower value than the specified value. The routines **symFindByNameAndType()** and **symByValueAndTypeFind()** allow the symbol type to be used as an additional criterion in the searches.

The routines **symFindByValue()** and **symFindByValueAndType()** are obsolete. They are replaced by the routines **symByValueFind()** and **symByValueAndTypeFind()**.

Symbols in the symbol table are hashed by name into a hash table for fast look-up by name, *e.g.*, by **symFindByName()**. The size of the hash table is specified during the creation of a symbol table. Look-ups by value, *e.g.*, **symByValueFind()**, must search the table linearly; these look-ups can thus be much slower.

The routine **symEach()** allows each symbol in the symbol table to be examined by a user-specified function.

Name clashes occur when a symbol added to a table is identical in name and type to a previously added symbol. Whether symbol tables can accept name clashes is set by a parameter when the symbol table is created with **symTblCreate()**. If name clashes are not allowed, **symAdd()** returns an error if there is an attempt to add a symbol with identical name and type. If name clashes are allowed, adding multiple symbols with the same name and type will be permitted. In such cases, **symFindByName()** will return the value most recently added, although all versions of the symbol can be found by **symEach()**.

The system symbol table (**sysSymTbl**) allows name clashes.

See the *VxWorks Programmer's Guide* for more information about configuration, initialization, and use of the system symbol table.

**INCLUDE FILES**    **symLib.h**

**SEE ALSO**        **loadLib**

---

## symSyncLib

**NAME**            **symSyncLib** – host/target symbol table synchronization

**ROUTINES**        **symSyncLibInit()** - initialize host/target symbol table synchronization  
**symSyncTimeoutSet()** - set WTX timeout

**DESCRIPTION**    This module provides host/target symbol table synchronization. With synchronization, every module or symbol added to the run-time system from either the target or host side can be seen by facilities on both the target and the host. Symbol table synchronization makes it possible to use host tools to debug application modules loaded with the target loader or from a target file system. To enable synchronization, two actions must be performed:

- 1 The module is initialized by **symSyncLibInit()**, which is called automatically when the configuration macro **INCLUDE\_SYM\_TBL\_SYNC** is defined.
- 2 The target server is launched with the **-s** option.

If synchronization is enabled, **symSyncLib** spawns a synchronization task on the target, **tSymSync**. This task behaves as a WTX tool and attaches itself to the target server. When the task starts, it synchronizes target and host symbol tables so that every module loaded on the target before the target server was started can be seen by the host tools. This feature is particularly useful if VxWorks is started with a target-based startup script before the target server has been launched.

The **tSymSync** task synchronizes new symbols that are added by either the target or the host tools. The task waits for synchronization events on two channels: a WTX event from the host or a message queue addition from the target.

The **tSymSync** task, like all WTX tools, must be able to connect to the WTX registry. To make the WTX registry accessible from the target, do one of the following:

- 1 Boot the target from a host on the same subnet as the registry.
- 2 Start the registry on the same host the target boots from.
- 3 Add the needed routes with **routeAdd()** calls, possibly in a startup script.

Neither the host tools nor the target loader wait for synchronization completion to return. To know when the synchronization is complete, you can wait for the corresponding event sent by the target server, or, if your target server was started with the **-V** option, it prints a message indicating synchronization has completed.

The event sent by the target server is of the following format:

```
SYNC_DONE syncType syncObj syncStatus
```

The following are examples of messages displayed by the target server indicating synchronization is complete:

```
Added target_modules          to target-server.....done  
Added ttTest.o.68k           to target.....done
```

If synchronization fails, the following message is displayed:

```
Added gopher.o              to target.....failed
```

This error generally means that synchronization of the corresponding module or symbol is no longer possible because it no longer exists in the original symbol table. If so, it will be followed by:

```
Removed gopher.o           from target.....failed
```

Failure can also occur if a timeout is reached. Call **symSyncTimeoutSet()** to modify the WTX timeout between the target synchronization task and the target server.

#### **LIMITATIONS**

Hardware: Because the synchronization task uses the WTX protocol to communicate with the target server, the target must include network facilities. Depending on how much synchronization is to be done (number of symbols to transfer), a reasonable throughput between the target server and target agent is required (the **wdbrpc** backend is recommended when large modules are to be loaded).

#### **PERFORMANCE**

The synchronization task requires some minor overhead in target routines **msgQSend()**, **loadModule()**, **symAdd()**, and **symRemove()**; however, if an application sends more than 15 synchronization events, it will fill the message queue and then need to wait for a synchronization event to be processed by **tSymSync**. Also, waiting for host synchronization events is done by polling; thus there may be some impact on performance if there are lower-priority tasks than **tSymSync**. If no more synchronization is needed, **tSymSync** can be suspended.

**KNOWN PROBLEM** Modules with undefined symbols that are loaded from the target are not synchronized; however, they are synchronized if they are loaded from the host.

**SEE ALSO** `tgtsvr`

---

## sysLib

**NAME** `sysLib` – system-dependent library

**ROUTINES**

- `sysClkConnect()` - connect a routine to the system clock interrupt
- `sysClkDisable()` - turn off system clock interrupts
- `sysClkEnable()` - turn on system clock interrupts
- `sysClkRateGet()` - get the system clock rate
- `sysClkRateSet()` - set the system clock rate
- `sysAuxClkConnect()` - connect a routine to the auxiliary clock interrupt
- `sysAuxClkDisable()` - turn off auxiliary clock interrupts
- `sysAuxClkEnable()` - turn on auxiliary clock interrupts
- `sysAuxClkRateGet()` - get the auxiliary clock rate
- `sysAuxClkRateSet()` - set the auxiliary clock rate
- `sysIntDisable()` - disable a bus interrupt level
- `sysIntEnable()` - enable a bus interrupt level
- `sysBusIntAck()` - acknowledge a bus interrupt
- `sysBusIntGen()` - generate a bus interrupt
- `sysMailboxConnect()` - connect a routine to the mailbox interrupt
- `sysMailboxEnable()` - enable the mailbox interrupt
- `sysNvRamGet()` - get the contents of non-volatile RAM
- `sysNvRamSet()` - write to non-volatile RAM
- `sysModel()` - return the model name of the CPU board
- `sysBspRev()` - return the BSP version and revision number
- `sysHwInit()` - initialize the system hardware
- `sysPhysMemTop()` - get the address of the top of memory
- `sysMemTop()` - get the address of the top of logical memory
- `sysToMonitor()` - transfer control to the ROM monitor
- `sysProcNumGet()` - get the processor number
- `sysProcNumSet()` - set the processor number
- `sysBusTas()` - test and set a location across the bus
- `sysScsiBusReset()` - assert the RST line on the SCSI bus (Western Digital WD33C93 only)
- `sysScsiInit()` - initialize an on-board SCSI port
- `sysScsiConfig()` - system SCSI configuration
- `sysLocalToBusAdrs()` - convert a local address to a bus address
- `sysBusToLocalAdrs()` - convert a bus address to a local address
- `sysSerialHwInit()` - initialize the BSP serial devices to a quiescent state

**sysLib**

- sysSerialHwInit2()** - connect BSP serial device interrupts
- sysSerialReset()** - reset all SIO devices to a quiet state
- sysSerialChanGet()** - get the **SIO\_CHAN** device associated with a serial channel
- sysNanoDelay()** - delay for specified number of nanoseconds

**DESCRIPTION**

This library provides board-specific routines.

---

**NOTE:** This is a generic reference entry for a BSP-specific library; this description contains general information only. For features and capabilities specific to the system library included in your BSP, see your BSP's reference entry for **sysLib**.

---

The file **sysLib.c** provides the board-level interface on which VxWorks and application code can be built in a hardware-independent manner. The functions addressed in this file include:

Initialization functions

- initialize the hardware to a known state
- identify the system
- initialize drivers, such as SCSI or custom drivers

Memory/address space functions

- get the on-board memory size
- make on-board memory accessible to external bus
- map local and bus address spaces
- enable/disable cache memory
- set/get nonvolatile RAM (NVRAM)
- define board's memory map (optional)
- virtual-to-physical memory map declarations for processors with MMUs

Bus interrupt functions

- enable/disable bus interrupt levels
- generate bus interrupts

Clock/timer functions

- enable/disable timer interrupts
- set the periodic rate of the timer

Mailbox/location monitor functions

- enable mailbox/location monitor interrupts for VME-based boards

The **sysLib** library does not support every feature of every board; a particular board may have various extensions to the capabilities described here. Conversely, some boards do not support every function provided by this library. Some boards provide some of the functions of this library by means of hardware switches, jumpers, or PALs, instead of software-controllable registers.

Typically, most functions in this library are not called by the user application directly. The configuration modules **usrConfig.c** and **bootConfig.c** are responsible for invoking the

routines at the appropriate time. Device drivers may use some of the memory mapping routines and bus functions.

**INCLUDE FILES**    **sysLib.h**

**SEE ALSO**        *VxWorks Programmer's Guide: Configuration and Build*, BSP-specific reference entry for **sysLib**

## tapeFsLib

<b>NAME</b>	<b>tapeFsLib</b> – tape sequential device file system library
<b>ROUTINES</b>	<b>tapeFsDevInit()</b> - associate a sequential device with tape volume functions <b>tapeFsInit()</b> - initialize the tape volume library <b>tapeFsReadyChange()</b> - notify <b>tapeFsLib</b> of a change in ready status <b>tapeFsVolUnmount()</b> - disable a tape device volume
<b>DESCRIPTION</b>	This library provides basic services for tape devices that do not use a standard file or directory structure on tape. The tape volume is treated much like a large file. The tape may either be read or written. However, there is no high-level organization of the tape into files or directories, which must be provided by a higher-level layer.

### USING THIS LIBRARY

The various routines provided by the VxWorks tape file system, or **tapeFs**, can be categorized into three broad groupings: general initialization, device initialization, and file system operation.

The **tapeFsInit()** routine is the principal general initialization function; it needs to be called only once, regardless of how many **tapeFs** devices are used.

To initialize devices, **tapeFsDevInit()** must be called for each **tapeFs** device.

Use of this library typically occurs through standard use of the I/O system routines **open()**, **close()**, **read()**, **write()** and **ioctl()**. Besides these standard I/O system operations, several routines are provided to inform the file system of changes in the system environment. The **tapeFsVolUnmount()** routine informs the file system that a particular device should be unmounted; any synchronization should be done prior to invocation of this routine, in preparation for a tape volume change. The **tapeFsReadyChange()** routine is used to inform the file system that a tape may have been swapped and that the next tape operation should first remount the tape. Information about a ready-change is also obtained from the driver using the **SEQ\_DEV** device structure. Note that **tapeFsVolUnmount()** and **tapeFsReadyChange()** should be called only after a file has been closed.

### INITIALIZATION OF THE FILE SYSTEM

Before any other routines in **tapeFsLib** can be used, **tapeFsInit()** must be called to initialize the library. This implementation of the tape file system assumes only one file descriptor per volume. However, this constraint can be changed in case a future implementation demands multiple file descriptors per volume.

During the **tapeFsInit()** call, the tape device library is installed as a driver in the I/O system driver table. The driver number associated with it is then placed in a global variable, **tapeFsDrvNum**.

To enable this initialization, define `INCLUDE_TAPEFS` in the BSP, or simply start using the tape file system with a call to `tapeFsDevInit()` and `tapeFsInit()` will be called automatically if it has not been called before.

#### DEFINING A TAPE DEVICE

To use this library for a particular device, the device structure used by the device driver must contain, as the very first item, a sequential device description structure (`SEQ_DEV`). The `SEQ_DEV` must be initialized before calling `tapeFsDevInit()`. The driver places in the `SEQ_DEV` structure the addresses of routines that it must supply: one that reads one or more blocks, one that writes one or more blocks, one that performs I/O control (`ioctl()`) on the device, one that writes file marks on a tape, one that rewinds the tape volume, one that reserves a tape device for use, one that releases a tape device after use, one that mounts/unmounts a volume, one that spaces forward or backwards by blocks or file marks, one that erases the tape, one that resets the tape device, and one that checks the status of the device. The `SEQ_DEV` structure also contains fields that describe the physical configuration of the device. For more information about defining sequential devices, see the *VxWorks Programmer's Guide: I/O System*.

#### INITIALIZATION OF THE DEVICE

The `tapeFsDevInit()` routine is used to associate a device with the `tapeFsLib` functions. The `volName` parameter expected by `tapeFsDevInit()` is a pointer to a name string which identifies the device. This string serves as the pathname for I/O operations which operate on the device and appears in the I/O system device table, which can be displayed using `iosDevShow()`.

The `pSeqDev` parameter expected by `tapeFsDevInit()` is a pointer to the `SEQ_DEV` structure describing the device and containing the addresses of the required driver functions.

The `pTapeConfig` parameter is a pointer to a `TAPE_CONFIG` structure that contains information specifying how the tape device should be configured. The configuration items are fixed/variable block size, rewind/no-rewind device, and number of file marks to be written. For more information about the `TAPE_CONFIG` structure, look at the header file `tapeFsLib.h`.

The syntax of the `tapeFsDevInit()` routine is as follows:

```
tapeFsDevInit
(
    char *          volName,      /* name to be used for volume */
    SEQ_DEV *      pSeqDev,      /* pointer to device descriptor */
    TAPE_CONFIG *  pTapeConfig /* pointer to tape config info */
)
```

When `tapeFsLib` receives a request from the I/O system, after `tapeFsDevInit()` has been called, it calls the device driver routines (whose addresses were passed in the `SEQ_DEV` structure) to access the device.

#### OPENING AND CLOSING A FILE

A tape volume is opened by calling the I/O system routine **open()**. A file can be opened only with the **O\_RDONLY** or **O\_WRONLY** flags. The **O\_RDWR** mode is not used by this library. A call to **open()** initializes the file descriptor buffer and state information, reserves the tape device, rewinds the tape device if it was configured as a rewind device, and mounts a volume. Once a tape volume has been opened, that tape device is reserved, disallowing any other system from accessing that device until the tape volume is closed. Also, the single file descriptor is marked “in use” until the file is closed, making sure that a file descriptor is not opened multiple times.

A tape device is closed by calling the I/O system routine **close()**. Upon a **close()** request, any unwritten buffers are flushed, the device is rewound (if it is a rewind device), and, finally, the device is released.

#### UNMOUNTING VOLUMES (CHANGING TAPES)

A tape volume should be unmounted before it is removed. When unmounting a volume, make sure that any open file is closed first. A tape may be unmounted by calling **tapeFsVolUnmount()** directly.

If a file is open, it is not correct to change the medium and continue with the same file descriptor still open. Since **tapeFs** assumes only one file descriptor per device, to reuse that device, the file must be closed and opened later for the new tape volume.

Before **tapeFsVolUnmount()** is called, the device should be synchronized by invoking the **ioctl()** **FIOSYNC** or **FIOFLUSH**. It is the responsibility of the higher-level layer to synchronize the tape file system before unmounting. Failure to synchronize the volume before unmounting may result in loss of data.

**IOCTL FUNCTIONS** The VxWorks tape sequential device file system supports the following **ioctl()** functions. The functions listed are defined in the header files **ioLib.h** and **tapeFsLib.h**.

##### FIOFLUSH

Writes all modified file descriptor buffers to the physical device.

```
status = ioctl (fd, FIOFLUSH, 0);
```

##### FIOSYNC

Performs the same function as **FIOFLUSH**.

##### FIOBLKSIZEGET

Returns the value of the block size set on the physical device. This value is compared against the **sd\_blkSize** value set in the **SEQ\_DEV** device structure.

##### FIOBLKSIZESET

Sets a specified block size value on the physical device and also updates the value in the **SEQ\_DEV** and **TAPE\_VOL\_DESC** structures, unless the supplied value is zero, in which case the device structures are updated but the device is not set to zero. This is because zero implies variable block operations, therefore the device block size is ignored.

**MTIOCTOP**

Allows use of the standard UNIX MTIO **ioctl** operations by means of the MTOP structure. The MTOP structure appears as follows:

```
typedef struct mtop
{
    short      mt_op;           /* operation */
    int        mt_count;       /* number of operations */
} MTOP;
```

Use these **ioctl()** operations as follows:

```
MTOP mtop;
mtop.mt_op = MTWEOF;
mtop.mt_count = 1;
status = ioctl (fd, MTIOCTOP, (int) &mtop);
```

The permissible values for **mt\_op** are:

**MTWEOF**

Writes an end-of-file record to tape. An end-of-file record is a file mark.

**MTFSF**

Forward space over a file mark and position the tape head in the gap between the file mark just skipped and the next data block. Any buffered data is flushed out to the tape if the tape is in write mode.

**MTBSF**

Backward space over a file mark and position the tape head in the gap preceding the file mark, that is, right before the file mark. Any buffered data is flushed out to the tape if the tape is in write mode.

**MTFSR**

Forward space over a data block and position the tape head in the gap between the block just skipped and the next block. Any buffered data is flushed out to the tape if the tape is in write mode.

**MTBSR**

Backward space over a data block and position the tape head right before the block just skipped. Any buffered data is flushed out to the tape if the tape is in write mode.

**MTREW**

Rewind the tape to the beginning of the medium. Any buffered data is flushed out to the tape if the tape is in write mode.

**MTOFFL**

Rewind and unload the tape. Any buffered data is flushed out to the tape if the tape is in write mode.

**MTNOP**

No operation, but check the status of the device, thus setting the appropriate

**tarLib**

SEQ\_DEV fields.

**MTRETEN**

Retention the tape. This command usually sets tape tension and can be used in either read or write mode. Any buffered data is flushed out to tape if the tape is in write mode.

**MTERASE**

Erase the entire tape and rewind it.

**MTEOM**

Position the tape at the end of the medium and unload the tape. Any buffered data is flushed out to the tape if the tape is in write mode.

**INCLUDE FILES**    **tapeFsLib.h**

**SEE ALSO**        **ioLib**, **iosLib**, *VxWorks Programmer's Guide: I/O System, Local File Systems*

---

## tarLib

**NAME**            **tarLib** – UNIX tar compatible library

**ROUTINES**        **tarExtract()** - extract all files from a tar formatted tape  
**tarArchive()** - archive named file/dir onto tape in tar format  
**tarToc()**        - display all contents of a tar formatted tape

**DESCRIPTION**    This library implements functions for archiving, extracting and listing of UNIX-compatible “tar” file archives. It can be used to archive and extract entire file hierarchies to/from archive files on local or remote disks, or directly to/from magnetic tapes.

**SEE ALSO**        **dosFsLib**

**CURRENT LIMITATIONS**

This Tar utility does not handle MS-DOS file attributes, when used in conjunction with the MS-DOS file system. The maximum subdirectory depth supported by this library is 16, while the total maximum path name that can be handled by tar is limited at 100 characters.

---

## taskArchLib

<b>NAME</b>	<b>taskArchLib</b> – architecture-specific task management routines
<b>ROUTINES</b>	<b>taskSRSet()</b> - set the task status register (68K, MIPS, x86) <b>taskSRInit()</b> - initialize the default task status register (MIPS)
<b>DESCRIPTION</b>	This library provides architecture-specific task management routines that set and examine architecture-dependent registers. For information about architecture-independent task management facilities, see the manual entry for <b>taskLib</b> .
	<b>NOTE:</b> There are no application-level routines in <b>taskArchLib</b> for SimSolaris, SimNT or SH.
<b>INCLUDE FILES</b>	regs.h, taskArchLib.h
<b>SEE ALSO</b>	taskLib

---

## taskHookLib

<b>NAME</b>	<b>taskHookLib</b> – task hook library
<b>ROUTINES</b>	<b>taskHookInit()</b> - initialize task hook facilities <b>taskCreateHookAdd()</b> - add a routine to be called at every task create <b>taskCreateHookDelete()</b> - delete a previously added task create routine <b>taskSwitchHookAdd()</b> - add a routine to be called at every task switch <b>taskSwitchHookDelete()</b> - delete a previously added task switch routine <b>taskDeleteHookAdd()</b> - add a routine to be called at every task delete <b>taskDeleteHookDelete()</b> - delete a previously added task delete routine
<b>DESCRIPTION</b>	This library provides routines for adding extensions to the VxWorks tasking facility. To allow task-related facilities to be added to the system without modifying the kernel, the kernel provides call-outs every time a task is created, switched, or deleted. The call-outs allow additional routines, or “hooks,” to be invoked whenever these events occur. The hook management routines below allow hooks to be dynamically added to and deleted from the current lists of create, switch, and delete hooks:  <b>taskCreateHookAdd()</b> and <b>taskCreateHookDelete()</b> Add and delete routines to be called when a task is created.

**taskSwitchHookAdd()** and **taskSwitchHookDelete()**

Add and delete routines to be called when a task is switched.

**taskDeleteHookAdd()** and **taskDeleteHookDelete()**

Add and delete routines to be called when a task is deleted.

This facility is used by **dbgLib** to provide task-specific breakpoints and single-stepping. It is used by **taskVarLib** for the “task variable” mechanism. It is also used by **fppLib** for floating-point coprocessor support.

---

**NOTE:** It is possible to have dependencies among task hook routines. For example, a delete hook may use facilities that are cleaned up and deleted by another delete hook. In such cases, the order in which the hooks run is important. VxWorks runs the create and switch hooks in the order in which they were added, and runs the delete hooks in reverse of the order in which they were added. Thus, if the hooks are added in “hierarchical” order, such that they rely only on facilities whose hook routines have already been added, then the required facilities will be initialized before any other facilities need them, and will be deleted after all facilities are finished with them.

---

VxWorks facilities guarantee this by having each facility’s initialization routine first call any prerequisite facility’s initialization routine before adding its own hooks. Thus, the hooks are always added in the correct order. Each initialization routine protects itself from multiple invocations, allowing only the first invocation to have any effect.

**INCLUDE FILES**

**taskHookLib.h**

**SEE ALSO**

**dbgLib**, **fppLib**, **taskLib**, **taskVarLib** *VxWorks Programmer’s Guide: Basic OS*

---

## taskHookShow

**NAME**

**taskHookShow** – task hook show routines

**ROUTINES**

**taskHookShowInit()** - initialize the task hook show facility  
**taskCreateHookShow()** - show the list of task create routines  
**taskSwitchHookShow()** - show the list of task switch routines  
**taskDeleteHookShow()** - show the list of task delete routines

**DESCRIPTION**

This library provides routines which summarize the installed kernel hook routines. There is one routine dedicated to the display of each type of kernel hook: task operation, task switch, and task deletion.

The routine **taskHookShowInit()** links the task hook show facility into the VxWorks system. It is called automatically when this show facility is configured into VxWorks using either of the following methods:

- If you use the configuration header files, define `INCLUDE_SHOW_ROUTINES` in `config.h`.
- If you use the Tornado project facility, select `INCLUDE_TASK_HOOK_SHOW`.

**INCLUDE FILES**     `taskHookLib.h`

**SEE ALSO**         `taskHookLib`, *VxWorks Programmer's Guide: Basic OS*

---

## taskInfo

**NAME**             `taskInfo` – task information library

**ROUTINES**        `taskOptionsSet()` - change task options  
`taskOptionsGet()` - examine task options  
`taskRegsGet()` - get a task's registers from the TCB  
`taskRegsSet()` - set a task's registers  
`taskName()` - get the name associated with a task ID  
`taskNameToId()` - look up the task ID associated with a task name  
`taskIdDefault()` - set the default task ID  
`taskIsReady()` - check if a task is ready to run  
`taskIsSuspended()` - check if a task is suspended  
`taskIdListGet()` - get a list of active task IDs

**DESCRIPTION**    This library provides a programmatic interface for obtaining task information.

Task information is crucial as a debugging aid and user-interface convenience during the development cycle of an application. The routines `taskOptionsGet()`, `taskRegsGet()`, `taskName()`, `taskNameToId()`, `taskIsReady()`, `taskIsSuspended()`, and `taskIdListGet()` are used to obtain task information. Three routines -- `taskOptionsSet()`, `taskRegsSet()`, and `taskIdDefault()` -- provide programmatic access to debugging features.

The chief drawback of using task information is that tasks may change their state between the time the information is gathered and the time it is utilized. Information provided by these routines should therefore be viewed as a snapshot of the system, and not relied upon unless the task is consigned to a known state, such as suspended.

Task management and control routines are provided by `taskLib`. Higher-level task information display routines are provided by `taskShow`.

**INCLUDE FILES**     `taskLib.h`

**SEE ALSO**         `taskLib`, `taskShow`, `taskHookLib`, `taskVarLib`, `semLib`, `kernelLib`, *VxWorks Programmer's Guide: Basic OS*

## taskLib

**NAME** taskLib – task management library

**ROUTINES**

- taskSpawn()** - spawn a task
- taskInit()** - initialize a task with a stack at a specified address
- taskActivate()** - activate a task that has been initialized
- exit()** - exit a task (ANSI)
- taskDelete()** - delete a task
- taskDeleteForce()** - delete a task without restriction
- taskSuspend()** - suspend a task
- taskResume()** - resume a task
- taskRestart()** - restart a task
- taskPrioritySet()** - change the priority of a task
- taskPriorityGet()** - examine the priority of a task
- taskLock()** - disable task rescheduling
- taskUnlock()** - enable task rescheduling
- taskSafe()** - make the calling task safe from deletion
- taskUnsafe()** - make the calling task unsafe from deletion
- taskDelay()** - delay a task from executing
- taskIdSelf()** - get the task ID of a running task
- taskIdVerify()** - verify the existence of a task
- taskTcb()** - get the task control block for a task ID

**DESCRIPTION** This library provides the interface to the VxWorks task management facilities. Task control services are provided by the VxWorks kernel, which is comprised of **kernelLib**, **taskLib**, **semLib**, **tickLib**, **msgQLib**, and **wdLib**. Programmatic access to task information and debugging features is provided by **taskInfo**. Higher-level task information display routines are provided by **taskShow**.

**TASK CREATION** Tasks are created with the general-purpose routine **taskSpawn()**. Task creation consists of the following: allocation of memory for the stack and task control block (**WIND\_TCB**), initialization of the **WIND\_TCB**, and activation of the **WIND\_TCB**. Special needs may require the use of the lower-level routines **taskInit()** and **taskActivate()**, which are the underlying primitives of **taskSpawn()**.

Tasks in VxWorks execute in the most privileged state of the underlying architecture. In a shared address space, processor privilege offers no protection advantages and actually hinders performance.

There is no limit to the number of tasks created in VxWorks, as long as sufficient memory is available to satisfy allocation requirements.

The routine **sp()** is provided in **usrLib** as a convenient abbreviation for spawning tasks. It calls **taskSpawn()** with default parameters.

- TASK DELETION** If a task exits its “main” routine, specified during task creation, the kernel implicitly calls **exit()** to delete the task. Tasks can be explicitly deleted with the **taskDelete()** or **exit()** routine.
- Task deletion must be handled with extreme care, due to the inherent difficulties of resource reclamation. Deleting a task that owns a critical resource can cripple the system, since the resource may no longer be available. Simply returning a resource to an available state is not a viable solution, since the system can make no assumption as to the state of a particular resource at the time a task is deleted.
- The solution to the task deletion problem lies in deletion protection, rather than overly complex deletion facilities. Tasks may be protected from unexpected deletion using **taskSafe()** and **taskUnsafe()**. While a task is safe from deletion, deleters will block until it is safe to proceed. Also, a task can protect itself from deletion by taking a mutual-exclusion semaphore created with the **SEM\_DELETE\_SAFE** option, which enables an implicit **taskSafe()** with each **semTake()**, and a **taskUnsafe()** with each **semGive()** (see **semMLib** for more information). Many VxWorks system resources are protected in this manner, and application designers may wish to consider this facility where dynamic task deletion is a possibility.
- The **sigLib** facility may also be used to allow a task to execute clean-up code before actually expiring.
- TASK CONTROL** Tasks are manipulated by means of an ID that is returned when a task is created. VxWorks uses the convention that specifying a task ID of **NULL** in a task control function signifies the calling task.
- The following routines control task state: **taskResume()**, **taskSuspend()**, **taskDelay()**, **taskRestart()**, **taskPrioritySet()**, and **taskRegsSet()**.
- TASK SCHEDULING** VxWorks schedules tasks on the basis of priority. Tasks may have priorities ranging from 0, the highest priority, to 255, the lowest priority. The priority of a task in VxWorks is dynamic, and an existing task’s priority can be changed using **taskPrioritySet()**.
- INCLUDE FILES** **taskLib.h**
- SEE ALSO** **taskInfo**, **taskShow**, **taskHookLib**, **taskVarLib**, **semLib**, **semMLib**, **kernelLib**, *VxWorks Programmer’s Guide: Basic OS*

## taskShow

<b>NAME</b>	<b>taskShow</b> – task show routines
<b>ROUTINES</b>	<b>taskShowInit()</b> - initialize the task show routine facility <b>taskInfoGet()</b> - get information about a task <b>taskShow()</b> - display task information from TCBS <b>taskRegsShow()</b> - display the contents of a task's registers <b>taskStatusString()</b> - get a task's status as a string
<b>DESCRIPTION</b>	<p>This library provides routines to show task-related information, such as register values, task status, <i>etc.</i></p> <p>The <b>taskShowInit()</b> routine links the task show facility into the VxWorks system. It is called automatically when this show facility is configured into VxWorks using either of the following methods:</p> <ul style="list-style-type: none"><li>– If you use the configuration header files, define <b>INCLUDE_SHOW_ROUTINES</b> in <b>config.h</b>.</li><li>– If you use the Tornado project facility, select <b>INCLUDE_TASK_SHOW</b>.</li></ul> <p>Task information is crucial as a debugging aid and user-interface convenience during the development cycle of an application. The routines <b>taskInfoGet()</b>, <b>taskShow()</b>, <b>taskRegsShow()</b>, and <b>taskStatusString()</b> are used to display task information.</p> <p>The chief drawback of using task information is that tasks may change their state between the time the information is gathered and the time it is utilized. Information provided by these routines should therefore be viewed as a snapshot of the system, and not relied upon unless the task is consigned to a known state, such as suspended.</p> <p>Task management and control routines are provided by <b>taskLib</b>. Programmatic access to task information and debugging features is provided by <b>taskInfo</b>.</p>
<b>INCLUDE FILES</b>	<b>taskLib.h</b>
<b>SEE ALSO</b>	<b>taskLib</b> , <b>taskInfo</b> , <b>taskHookLib</b> , <b>taskVarLib</b> , <b>semLib</b> , <b>kernelLib</b> , <i>VxWorks Programmer's Guide: Basic OS, Target Shell, Tornado User's Guide: Shell</i>

---

## taskVarLib

<b>NAME</b>	<b>taskVarLib</b> – task variables support library
<b>ROUTINES</b>	<b>taskVarInit()</b> - initialize the task variables facility <b>taskVarAdd()</b> - add a task variable to a task <b>taskVarDelete()</b> - remove a task variable from a task <b>taskVarGet()</b> - get the value of a task variable <b>taskVarSet()</b> - set the value of a task variable <b>taskVarInfo()</b> - get a list of task variables of a task
<b>DESCRIPTION</b>	<p>VxWorks provides a facility called “task variables,” which allows 4-byte variables to be added to a task’s context, and the variables’ values to be switched each time a task switch occurs to or from the calling task. Typically, several tasks declare the same variable (4-byte memory location) as a task variable and treat that memory location as their own private variable. For example, this facility can be used when a routine must be spawned more than once as several simultaneous tasks.</p> <p>The routines <b>taskVarAdd()</b> and <b>taskVarDelete()</b> are used to add or delete a task variable. The routines <b>taskVarGet()</b> and <b>taskVarSet()</b> are used to get or set the value of a task variable.</p> <hr/> <p><b>NOTE:</b> If you are using task variables in a task delete hook (see <b>taskHookLib</b>), refer to the manual entry for <b>taskVarInit()</b> for warnings on proper usage.</p> <hr/>
<b>INCLUDE FILES</b>	<b>taskVarLib.h</b>
<b>SEE ALSO</b>	<b>taskHookLib</b> , <i>VxWorks Programmer’s Guide: Basic OS</i>

---

## tcpShow

<b>NAME</b>	<b>tcpShow</b> – TCP information display routines
<b>ROUTINES</b>	<b>tcpShowInit()</b> - initialize TCP show routines <b>tcpDebugShow()</b> - display debugging information for the TCP protocol <b>tcpstatShow()</b> - display all statistics for the TCP protocol
<b>DESCRIPTION</b>	<p>This library provides routines to show TCP related statistics.</p> <p>Interpreting these statistics requires detailed knowledge of Internet network protocols. Information on these protocols can be found in the following books:</p>

## **telnetdLib**

*TCP/IP Illustrated Volume II, The Implementation*, by Richard Stevens

*The Design and Implementation of the 4.4 BSD UNIX Operating System*, by Leffler, McKusick, Karels and Quarterman

The **tcpShowInit()** routine links the TCP show facility into the VxWorks system. This is performed automatically if **INCLUDE\_TCP\_SHOW** is defined.

**SEE ALSO**      **netLib, netShow**

---

## **telnetdLib**

**NAME**            **telnetdLib** – server library

**ROUTINES**        **telnetdInit()** - initialize the **telnet** services  
**telnetdParserSet()** - specify a command interpreter for **telnet** sessions  
**telnetdStart()** - initialize the **telnet** services  
**telnetdExit()** - close an active **telnet** session  
**telnetdStaticTaskInitializationGet()** - report whether tasks were pre-started by **telnetd**

**DESCRIPTION**    The **telnet** protocol enables users on remote systems to login to VxWorks.

This library implements a **telnet** server which accepts remote **telnet** login requests and transfers input and output data between a command interpreter and the remote user. The default configuration redirects the input and output from the VxWorks shell if available. The **telnetdParserSet()** routine allows the installation of an alternative command interpreter to handle the remote input and provide the output responses. If **INCLUDE\_SHELL** is not defined, installing a command interpreter is required.

The **telnetdInit()** routine initializes the **telnet** service when **INCLUDE\_TELNET** is defined. If **INCLUDE\_SHELL** is also defined, the **telnetdStart()** routine automatically starts the server. Client sessions will connect to the shell, which only supports one client at a time.

### **VXWORKS AE PROTECTION DOMAINS**

Under VxWorks AE, the **telnet** server runs within the kernel protection domain only. This restriction does not apply under non-AE versions of VxWorks.

**INCLUDE FILES**    **telnetLib.h**

**SEE ALSO**         **rlogLib**

---

## tffsConfig

<b>NAME</b>	<b>tffsConfig</b> – TrueFFS configuration file for VxWorks
<b>ROUTINES</b>	<b>tffsShowAll()</b> - show device information on all socket interfaces <b>tffsShow()</b> - show device information on a specific socket interface <b>tffsBootImagePut()</b> - write to the boot-image region of the flash device
<b>DESCRIPTION</b>	<p>This source file, with the help of <b>sysTffs.c</b>, configures TrueFFS for VxWorks. The functions defined here are generic to all BSPs. To include these functions in the BSP-specific module, the BSP's <b>sysTffs.c</b> file includes this file. Within the <b>sysTffs.c</b> file, define statements determine which functions from the <b>tffsConfig.c</b> file are ultimately included in TrueFFS.</p> <p>The only externally callable routines defined in this file are <b>tffsShow()</b>, <b>tffsShowAll()</b>, and <b>tffsBootImagePut()</b>. You can exclude the show utilities if you edit <b>config.h</b> and undefine <b>INCLUDE_SHOW_ROUTINES</b>. You can exclude <b>tffsBootImagePut()</b> if you edit <b>sysTffs.c</b> and undefine <b>INCLUDE_TFFS_BOOT_IMAGE</b>. (If you find these utilities are missing and you want them included, edit <b>config.h</b> and define <b>INCLUDE_SHOW_ROUTINES</b> and <b>INCLUDE_TFFS_BOOT_IMAGE</b>.)</p> <p>If you wish to include only the TrueFFS specific show routines you could define <b>INCLUDE_TFFS_SHOW</b> instead of <b>INCLUDE_SHOW_ROUTINES</b> in <b>config.h</b>.</p> <p>However, for the most part, these externally callable routines are only a small part of the TrueFFS configuration needs handled by this file. The routines internal to this file make calls into the MTDs and translation layer modules of TrueFFS. At link time, resolving the symbols associated with these calls pulls MTD and translation layer modules into VxWorks.</p> <p>However, each of these calls to the MTDs and the translation layer modules is only conditionally included. The constants that control the includes are defined in <b>sysTffs.c</b>. To exclude an MTD or translation layer module, you edit <b>sysTffs.c</b>, undefine the appropriate constant, and rebuild <b>sysTffs.o</b>. These constants are described in the reference entry for <b>sysTffs</b>.</p>
<b>INCLUDE FILES</b>	<b>stdcomp.h</b>

---

## tffsDrv

<b>NAME</b>	<b>tffsDrv</b> – TrueFFS interface for VxWorks
<b>ROUTINES</b>	<b>tffsDrv()</b> - initialize the TrueFFS system <b>tffsDevCreate()</b> - create a TrueFFS block device suitable for use with dosFs <b>tffsDevOptionsSet()</b> - set TrueFFS volume options <b>tffsDevFormat()</b> - format a flash device for use with TrueFFS <b>tffsRawio()</b> - low level I/O access to flash components
<b>DESCRIPTION</b>	<p>This module defines the routines that VxWorks uses to create a TrueFFS block device. Using this block device, dosFs can access a board-resident flash memory array or a flash memory card (in the PCMCIA slot) just as if it was a standard disk drive. Also defined in this file are functions that you can use to format the flash medium, as well as well as functions that handle the low-level I/O to the device.</p> <p>To include TrueFFS for Tornado in a VxWorks image, you must edit your BSP's <b>config.h</b> and define <b>INCLUDE_TFFS</b>, or, for some hardware, <b>INCLUDE_PCMCIA</b>. If you define <b>INCLUDE_TFFS</b>, this configures <b>usrRoot()</b> to call <b>tffsDrv()</b>. If you defined <b>INCLUDE_PCMCIA</b>, the call to <b>tffsDrv()</b> is made from <b>pccardTffsEnabler()</b>. The call to <b>tffsDrv()</b> sets up the structures, global variables, and mutual exclusion semaphore needed to manage TrueFFS. This call to <b>tffsDrv()</b> also registers socket component drivers for each flash device found attached to the target.</p> <p>These socket component drivers are not quite block devices, but they are an essential layer within TrueFFS. Their function is to manage the hardware interface to the flash device, and they are intelligent enough to handle formatting and raw I/O requests to the flash device. The other two layers within TrueFFS are known as the translation layer and the MTD (the Memory Technology Driver). The translation layer of TrueFFS implements the error recover and wear-leveling features of TrueFFS. The MTD implements the low-level programming (map, read, write, and erase) of the flash medium.</p> <p>To implement the socket layer, each BSP that supports TrueFFS includes a <b>sysTffs.c</b> file. This file contains the code that defines the socket component driver. This file also contains a set of defines that you can use to configure which translation layer modules and MTDs are included in TrueFFS. Which translation layer modules and MTDs you should include depends on which types of flash devices you need to support. Currently, there are three basic flash memory technologies, NAND-based, NOR-based, and SSFDC. Within <b>sysTffs.c</b>, define:</p> <p><b>INCLUDE_TL_NFTL</b> To include the NAND-based translation layer module.</p> <p><b>INCLUDE_TL_FTL</b> To include the NOR-based translation layer module.</p>

**INCLUDE\_TL\_SSFDC**

To include the SSFDC-appropriate translation layer module.

To support these different technologies, TrueFFS ships with three different implementations of the translation layer. Optionally, TrueFFS can include all three modules. TrueFFS later binds the appropriate translation layer module to the flash device when it registers a socket component driver for the device.

Within these three basic flash device categories there are still other differences (largely manufacturer-specific). These differences have no impact on the translation layer. However, they do make a difference for the MTD. Thus, TrueFFS ships with eight different MTDs that can support a variety of flash devices from Intel, Sharp, Samsung, National, Toshiba, AMD, and Fujitsu. Within `sysTffs.c`, define:

**INCLUDE\_MTD\_I28F016**

For Intel 28f016 flash devices.

**INCLUDE\_MTD\_I28F008**

For Intel 28f008 flash devices.

**INCLUDE\_MTD\_I28F008\_BAJA**

For Intel 28f008 flash devices on the Heurikon Baja 4000.

**INCLUDE\_MTD\_AMD**

For AMD, Fujitsu: 29F0{40,80,16} 8-bit flash devices.

**INCLUDE\_MTD\_CDSN**

For Toshiba, Samsung: NAND CDSN flash devices.

**INCLUDE\_MTD\_DOC2**

For Toshiba, Samsung: NAND DOC flash devices.

**INCLUDE\_MTD\_CFISCS**

For CFI/SCS flash devices.

**INCLUDE\_MTD\_WAMD**

For AMD, Fujitsu 29F0{40,80,16} 16-bit flash devices.

The socket component driver and the MTDs are provided in source form. If you need to write your own socket driver or MTD, use these working drivers as a model for your own.

**EXTERNALLY CALLABLE ROUTINES**

Most of the routines defined in this file are accessible through the I/O system only. However, four routines are callable externally. These are: `tffsDrv()`, `tffsDevCreate()`, `tffsDevFormat()`, and `tffsRawio()`.

The first routine called from this library must be `tffsDrv()`. Call this routine exactly once. Normally, this is handled automatically for you from within `usrRoot()`, if `INCLUDE_TFFS` is defined, or from within `pccardTffsEnabler()`, if `INCLUDE_PCMCIA` is defined.

Internally, this call to `tffsDrv()` registers socket component drivers for all the flash devices connected to your system. After registering a socket component driver for the device,

**tftpLib**

TrueFFS can support calls to **tffsDevFormat()** or **tffsRawio()**. However, before you can mount dosFs on the flash device, you must call **tffsDevCreate()**. This call creates a block device on top of the socket component driver, but does not mount dosFs on the device. Because mounting dosFs on the device is what you will want to do most of the time, the **sysTffs.c** file defines a helper function, **usrTffsConfig()**. Internally, this function calls **tffsDevCreate()** and then does everything necessary (such as calling the **dosFsDevInit()** routine) to mount dosFs on the resulting block device.

**LOW LEVEL I/O** Normally, you should handle your I/O to the flash device using dosFs. However, there are situations when that level of indirection is a problem. To handle such situations, this library defines **tffsRawio()**. Using this function, you can bypass both dosFs and the TrueFFS translation services to program the flash medium directly.

However, you should not try to program the flash device directly unless you are intimately familiar with the physical limits of your flash device as well as with how TrueFFS formats the flash medium. Otherwise you risk not only corrupting the medium entirely but permanently damaging the flash device.

If all you need to do is write a boot image to the flash device, use the **tffsBootImagePut()** utility instead of **tffsRawio()**. This function provides safer access to the flash medium.

**IOCTL** This driver responds to all ioctl codes by setting a global error flag. Do not attempt to format a flash drive using ioctl calls.

**INCLUDE FILES** **tffsDrv.h, fatlite.h**

---

## tftpLib

**NAME** **tftpLib** – Trivial File Transfer Protocol server library

**ROUTINES** **tftpInit()** - initialize the TFTP server task  
**tftpTask()** - TFTP server daemon task  
**tftpDirectoryAdd()** - add a directory to the access list  
**tftpDirectoryRemove()** - delete a directory from the access list

**DESCRIPTION** This library implements the VxWorks Trivial File Transfer Protocol (TFTP) server module. The server can respond to both read and write requests. It is started by a call to **tftpInit()**.

The server has access to a list of directories that can either be provided in the initial call to **tftpInit()** or changed dynamically using the **tftpDirectoryAdd()** and **tftpDirectoryRemove()** calls. Requests for files not in the directory trees specified in the access list will be rejected, unless the list is empty, in which case all requests will be

allowed. By default, the access list contains the directory given in the global variable **ftplibDirectory**. It is possible to remove the default by calling **ftplibDirectoryRemove()**.

For specific information about the TFTP protocol, see RFC 783, "TFTP Protocol."

#### VXWORKS AE PROTECTION DOMAINS

Under VxWorks AE, you can run the TFTP server in the kernel protection domain only. This restriction does not apply under non-AE versions of VxWorks.

**INCLUDE FILES** **ftplib.h, tftplib.h**

**SEE ALSO** **ftplib**, RFC 783 "TFTP Protocol"

---

## ftplib

**NAME** **ftplib** – Trivial File Transfer Protocol (TFTP) client library

**ROUTINES** **ftplibXfer()** - transfer a file via TFTP using a stream interface  
**ftplibCopy()** - transfer a file via TFTP  
**ftplibInit()** - initialize a TFTP session  
**ftplibModeSet()** - set the TFTP transfer mode  
**ftplibPeerSet()** - set the TFTP server address  
**ftplibPut()** - put a file to a remote system  
**ftplibGet()** - get a file from a remote system  
**ftplibInfoShow()** - get TFTP status information  
**ftplibQuit()** - quit a TFTP session  
**ftplibSend()** - send a TFTP message to the remote system

**DESCRIPTION** This library implements the VxWorks Trivial File Transfer Protocol (TFTP) client library. TFTP is a simple file transfer protocol (hence the name "trivial") implemented over UDP. TFTP was designed to be small and easy to implement. Therefore, it is limited in functionality in comparison with other file transfer protocols, such as FTP. TFTP provides only the read/write capability to and from a remote server.

TFTP provides no user authentication. Therefore, the remote files must have "loose" permissions before requests for file access will be granted by the remote TFTP server. This means that the files to be read must be publicly readable, and files to be written must exist and be publicly writable). Some TFTP servers offer a secure option (-s) that specifies a directory where the TFTP server is rooted. Refer to the host manuals for more information about a particular TFTP server.

#### HIGH-LEVEL INTERFACE

The **ftplib** library has two levels of interface. The tasks **ftplibXfer()** and **ftplibCopy()**

operate at the highest level and are the main call interfaces. The **tftpXfer()** routine provides a stream interface to TFTP. That is, it spawns a task to perform the TFTP transfer and provides a descriptor from which data can be transferred interactively. The **tftpXfer()** interface is similar to **ftpXfer()** in **ftpLib**. The **tftpCopy()** routine transfers a remote file to or from a passed file (descriptor).

#### LOW-LEVEL INTERFACE

The lower-level interface is made up of various routines that act on a TFTP session. Each TFTP session is defined by a TFTP descriptor. These routines include:

- tftpInit()** to initialize a session;
- tftpModeSet()** to set the transfer mode;
- tftpPeerSet()** to set a peer/server address;
- tftpPut()** to put a file to the remote system;
- tftpGet()** to get file from remote system;
- tftpInfoShow()** to show status information; and
- tftpQuit()** to quit a TFTP session.

#### EXAMPLE

The following code provides an example of how to use the lower-level routines. It implements roughly the same function as **tftpCopy()**.

```
char *      pHost;
int         port;
char *      pFilename;
char *      pCommand;
char *      pMode;
int         fd;
TFTP_DESC * pTftpDesc;
int         status;
if ((pTftpDesc = tftpInit ()) == NULL)
    return (ERROR);
if ((tftpPeerSet (pTftpDesc, pHost, port) == ERROR) ||
    (tftpModeSet (pTftpDesc, pMode) == ERROR))
{
    (void) tftpQuit (pTftpDesc);
    return (ERROR);
}
if (strcmp (pCommand, "get") == 0)
{
    status = tftpGet (pTftpDesc, pFilename, fd, TFTP_CLIENT);
}
else if (strcmp (pCommand, "put") == 0)
{
    status = tftpPut (pTftpDesc, pFilename, fd, TFTP_CLIENT);
}
else
```

```

    {
        errno = S_tftpLib_INVALID_COMMAND;
        status = ERROR;
    }
    (void) tftpQuit (pTftpDesc);

```

To use this feature, include the following component: `INCLUDE_TFTP_CLIENT`

**INCLUDE FILES**     `tftpLib.h`

**SEE ALSO**         `tftpdLib`

---

## tickLib

**NAME**             `tickLib` – clock tick support library

**ROUTINES**        `tickAnnounce()` - announce a clock tick to the kernel  
                   `tickSet()` - set the value of the kernel’s tick counter  
                   `tickGet()` - get the value of the kernel’s tick counter

**DESCRIPTION**    This library is the interface to the VxWorks kernel routines that announce a clock tick to the kernel, get the current time in ticks, and set the current time in ticks.

Kernel facilities that rely on clock ticks include `taskDelay()`, `wdStart()`, `kernelTimeslice()`, and semaphore timeouts. In each case, the specified timeout is relative to the current time, also referred to as “time to fire.” Relative timeouts are not affected by calls to `tickSet()`, which only changes absolute time. The routines `tickSet()` and `tickGet()` keep track of absolute time in isolation from the rest of the kernel.

Time-of-day clocks or other auxiliary time bases are preferable for lengthy timeouts of days or more. The accuracy of such time bases is greater, and some external time bases even calibrate themselves periodically.

**INCLUDE FILES**     `tickLib.h`

**SEE ALSO**         `kernelLib`, `taskLib`, `semLib`, `wdLib`, *VxWorks Programmer’s Guide: Basic OS*

## timerLib

<b>NAME</b>	<b>timerLib</b> – timer library (POSIX)
<b>ROUTINES</b>	<p><b>timer_cancel()</b> - cancel a timer</p> <p><b>timer_connect()</b> - connect a user routine to the timer signal</p> <p><b>timer_create()</b> - allocate a timer using the specified clock for a timing base (POSIX)</p> <p><b>timer_delete()</b> - remove a previously created timer (POSIX)</p> <p><b>timer_gettime()</b> - get the remaining time before expiration and the reload value (POSIX)</p> <p><b>timer_getoverrun()</b> - return the timer expiration overrun (POSIX)</p> <p><b>timer_settime()</b> - set the time until the next expiration and arm timer (POSIX)</p> <p><b>nanosleep()</b> - suspend the current task until the time interval elapses (POSIX)</p> <p><b>sleep()</b> - delay for a specified amount of time</p> <p><b>alarm()</b> - set an alarm clock for delivery of a signal</p>
<b>DESCRIPTION</b>	<p>This library provides a timer interface, as defined in the IEEE standard, POSIX 1003.1b.</p> <p>Timers are mechanisms by which tasks signal themselves after a designated interval. Timers are built on top of the clock and signal facilities. The clock facility provides an absolute time-base. Standard timer functions simply consist of creation, deletion and setting of a timer. When a timer expires, <b>sigaction()</b> (see <b>sigLib</b>) must be in place in order for the user to handle the event. The “high resolution sleep” facility, <b>nanosleep()</b>, allows sub-second sleeping to the resolution of the clock.</p> <p>The <b>clockLib</b> library should be installed and <b>clock_settime()</b> set before the use of any timer routines.</p>
<b>ADDITIONS</b>	<p>Two non-POSIX functions are provided for user convenience:</p> <ul style="list-style-type: none"> <li>– <b>timer_cancel()</b> quickly disables a timer by calling <b>timer_settime()</b>.</li> <li>– <b>timer_connect()</b> easily hooks up a user routine by calling <b>sigaction()</b>.</li> </ul>
<b>CLARIFICATIONS</b>	<p>The task creating a timer with <b>timer_create()</b> will receive the signal no matter which task actually arms the timer.</p> <p>When a timer expires and the task has previously exited, <b>logMsg()</b> indicates the expected task is not present. Similarly, <b>logMsg()</b> indicates when a task arms a timer without installing a signal handler. Timers may be armed but not created or deleted at interrupt level.</p>
<b>IMPLEMENTATION</b>	<p>The actual clock resolution is hardware-specific and in many cases is 1/60th of a second. This is less than <b>_POSIX_CLOCKRES_MIN</b>, which is defined as 20 milliseconds (1/50th of a second).</p>
<b>INCLUDE FILES</b>	<b>timers.h</b>
<b>SEE ALSO</b>	<b>clockLib</b> , <b>sigaction()</b> , POSIX 1003.1b documentation, <i>VxWorks Programmer’s Guide: Basic OS</i>

---

## timexLib

**NAME** **timexLib** – execution timer facilities

**ROUTINES**

- timexInit()** - include the execution timer library
- timexClear()** - clear the list of function calls to be timed
- timexFunc()** - specify functions to be timed
- timexHelp()** - display synopsis of execution timer facilities
- timex()** - time a single execution of a function or functions
- timexN()** - time repeated executions of a function or group of functions
- timexPost()** - specify functions to be called after timing
- timexPre()** - specify functions to be called prior to timing
- timexShow()** - display the list of function calls to be timed

**DESCRIPTION**

This library contains routines for timing the execution of programs, individual functions, and groups of functions. The VxWorks system clock is used as a time base. Functions that have a short execution time relative to this time base can be called repeatedly to establish an average execution time with an acceptable percentage of error.

Up to four functions can be specified to be timed as a group. Additionally, sets of up to four functions can be specified as pre- or post-timing functions, to be executed before and after the timed functions. The routines **timexPre()** and **timexPost()** are used to specify the pre- and post-timing functions, while **timexFunc()** specifies the functions to be timed.

The routine **timex()** is used to time a single execution of a function or group of functions. If called with no arguments, **timex()** uses the functions in the lists created by calls to **timexPre()**, **timexPost()**, and **timexFunc()**. If called with arguments, **timex()** times the function specified, instead of the previous list. The routine **timexN()** works in the same manner as **timex()** except that it iterates the function calls to be timed.

**EXAMPLES**

The routine **timex()** can be used to obtain the execution time of a single routine:

```
-> timex myFunc, myArg1, myArg2, ...
```

The routine **timexN()** calls a function repeatedly until a 2% or better tolerance is obtained:

```
-> timexN myFunc, myArg1, myArg2, ...
```

The routines **timexPre()**, **timexPost()**, and **timexFunc()** are used to specify a list of functions to be executed as a group:

```
-> timexPre 0, myPreFunc1, preArg1, preArg2, ...
-> timexPre 1, myPreFunc2, preArg1, preArg2, ...
-> timexFunc 0, myFunc1, myArg1, myArg2, ...
-> timexFunc 1, myFunc2, myArg1, myArg2, ...
-> timexFunc 2, myFunc3, myArg1, myArg2, ...
-> timexPost 0, myPostFunc, postArg1, postArg2, ...
```

The list is executed by calling **timex()** or **timexN()** without arguments:

```
-> timex
```

or:

```
-> timexN
```

In this example, *myPreFunc1* and *myPreFunc2* are called with their respective arguments. *myFunc1*, *myFunc2*, and *myFunc3* are then called in sequence and timed. If **timexN()** was used, the sequence is called repeatedly until a 2% or better error tolerance is achieved. Finally, *myPostFunc* is called with its arguments. The timing results are reported after all post-timing functions are called.

---

**NOTE:** The timings measure the execution time of the routine body, without the usual subroutine entry and exit code (usually LINK, UNLINK, and RTS instructions). Also, the time required to set up the arguments and call the routines is not included in the reported times. This is because these timing routines automatically calibrate themselves by timing the invocation of a null routine, and thereafter subtracting that constant overhead.

---

**INCLUDE FILES**     **timexLib.h**

**SEE ALSO**         **spyLib**

---

## trgLib

**NAME**             **trgLib** – trigger events control library

**ROUTINES**        **trgLibInit()** - initialize the triggering library  
**trgWorkQReset()** - resets the trigger work queue task and queue  
**trgAdd()** - add a new trigger to the trigger list  
**trgDelete()** - delete a trigger from the trigger list  
**trgOn()** - set triggering on  
**trgOff()** - set triggering off  
**trgEnable()** - enable a trigger  
**trgDisable()** - turn a trigger off  
**trgChainSet()** - chains two triggers  
**trgEvent()** - trigger a user-defined event

**DESCRIPTION**     This library provides the interface for triggering events. The routines provide tools for creating, deleting, and controlling triggers. However, in most cases it is preferable to use the GUI to create and manage triggers, since all order and dependency factors are automatically accounted for there.

The event types are defined as in WindView. Triggering and WindView share the same instrumentation points. Furthermore, one of the main uses of triggering is to start and stop WindView instrumentation. Triggering is started by the routine **trgOn()**, which sets the shared variable **evtAction**. Once the variable is set, when an instrumented point is hit, **trgCheck()** is called. The routine looks for triggers that apply to this event. The routine **trgOff()** stops triggering. The routine **trgEnable()** enables a specific trigger that was previously disabled with **trgDisable()**. (At creation time all triggers are enabled by default.) This routine also checks the number of triggers currently enabled, and when this is zero, it turns triggering off.

---

**NOTE:** It is important to create a trigger before calling **trgOn()**. **trgOn()** checks the trigger list to see if there is at least one trigger there, and if not, it exits without setting **evtAction**.

---

**INCLUDE FILES**     **trgLibP.h**

**SEE ALSO**         *WindView User's Guide*

---

## trgShow

**NAME**             **trgShow** – trigger show routine

**ROUTINES**        **trgShowInit()** - initialize the trigger show facility  
**trgShow()** - show trigger information

**DESCRIPTION**    This library provides routines to show event triggering information, such as list of triggers, associated actions, trigger states, and so on.

The routine **trgShowInit()** links the triggering show facility into the VxWorks system. It is called automatically when **INCLUDE\_TRIGGER\_SHOW** is defined.

**SEE ALSO**         **trgLib**

## ttyDrv

<b>NAME</b>	<b>ttyDrv</b> – provide terminal device access to serial channels
<b>ROUTINES</b>	<b>ttyDrv()</b> - initialize the <i>tty</i> driver <b>ttyDevCreate()</b> - create a VxWorks device for a serial channel
<b>DESCRIPTION</b>	<p>This library provides the OS-dependent functionality of a serial device, including canonical processing and the interface to the VxWorks I/O system.</p> <p>The BSP provides “raw” serial channels which are accessed via an <b>SIO_CHAN</b> data structure. These raw devices provide only low level access to the devices to send and receive characters. This library builds on that functionality by allowing the serial channels to be accessed via the VxWorks I/O system using the standard read/write interface. It also provides the canonical processing support of <b>tyLib</b>.</p> <p>The routines in this library are typically called by <b>usrRoot()</b> in <b>usrConfig.c</b> to create VxWorks serial devices at system startup time.</p>
<b>INCLUDE FILES</b>	<b>tyLib.h</b>
<b>SEE ALSO</b>	<b>tyLib</b> , <b>sioLib.h</b>

---

## tyLib

<b>NAME</b>	<b>tyLib</b> – <i>tty</i> driver support library
<b>ROUTINES</b>	<b>tyDevInit()</b> - initialize the <i>tty</i> device descriptor <b>tyDevRemove()</b> - remove the <i>tty</i> device descriptor <b>tyAbortFuncSet()</b> - set the abort function <b>tyAbortSet()</b> - change the abort character <b>tyBackspaceSet()</b> - change the backspace character <b>tyDeleteLineSet()</b> - change the line-delete character <b>tyEOFSet()</b> - change the end-of-file character <b>tyMonitorTrapSet()</b> - change the trap-to-monitor character <b>tyIoctl()</b> - handle device control requests <b>tyWrite()</b> - do a task-level write for a <i>tty</i> device <b>tyRead()</b> - do a task-level read for a <i>tty</i> device <b>tyITx()</b> - interrupt-level output <b>tyIRd()</b> - interrupt-level input

**DESCRIPTION** This library provides routines used to implement drivers for serial devices. It provides all the necessary device-independent functions of a normal serial channel, including:

- ring buffering of input and output
- raw mode
- optional line mode with backspace and line-delete functions
- optional processing of X-on/X-off
- optional RETURN/LINEFEED conversion
- optional echoing of input characters
- optional stripping of the parity bit from 8-bit input
- optional special characters for shell abort and system restart

Most of the routines in this library are called only by device drivers. Functions that normally might be called by an application or interactive user are the routines to set special characters, **ty...Set()**.

#### USE IN SERIAL DEVICE DRIVERS

Each device that uses **tyLib** is described by a data structure of type **TY\_DEV**. This structure begins with an I/O system device header so that it can be added directly to the I/O system's device list. A driver calls **tyDevInit()** to initialize a **TY\_DEV** structure for a specific device and then calls **iosDevAdd()** to add the device to the I/O system.

The call to **tyDevInit()** takes three parameters: the pointer to the **TY\_DEV** structure to initialize, the desired size of the read and write ring buffers, and the address of a transmitter start-up routine. This routine will be called when characters are added for output and the transmitter is idle. Thereafter, the driver can call the following routines to perform the usual device functions:

#### **tyRead()**

user read request to get characters that have been input

#### **tyWrite()**

user write request to put characters to be output

#### **tyIoctl()**

user I/O control request

#### **tyIRd()**

interrupt-level routine to get an input character

#### **tyITx()**

interrupt-level routine to deliver the next output character

Thus, **tyRead()**, **tyWrite()**, and **tyIoctl()** are called from the driver's read, write, and I/O control functions. The routines **tyIRd()** and **tyITx()** are called from the driver's interrupt handler in response to receive and transmit interrupts, respectively.

Examples of using **tyLib** in a driver can be found in the source file(s) included by **tyCoDrv**. Source files are located in **src/drv/serial**.

#### TTY OPTIONS

A full range of options affects the behavior of *tty* devices. These options are selected by setting bits in the device option word using the **FIOSETOPTIONS** function in the **ioctl()** routine (see *I/O Control Functions* below for more information). The following is a list of available options. The options are defined in the header file **ioLib.h**.

##### OPT\_LINE

Selects line mode. A *tty* device operates in one of two modes: raw mode (unbuffered) or line mode. Raw mode is the default. In raw mode, each byte of input from the device is immediately available to readers, and the input is not modified except as directed by other options below. In line mode, input from the device is not available to readers until a NEWLINE character is received, and the input may be modified by backspace, line-delete, and end-of-file special characters.

##### OPT\_ECHO

Causes all input characters to be echoed to the output of the same channel. This is done simply by putting incoming characters in the output ring as well as the input ring. If the output ring is full, the echoing is lost without affecting the input.

##### OPT\_CRMOD

C language conventions use the NEWLINE character as the line terminator on both input and output. Most terminals, however, supply a RETURN character when the return key is hit, and require both a RETURN and a LINEFEED character to advance the output line. This option enables the appropriate translation: NEWLINES are substituted for input RETURN characters, and NEWLINES in the output file are automatically turned into a RETURN-LINEFEED sequence.

##### OPT\_TANDEM

Causes the driver to generate and respond to the special flow control characters CTRL-Q and CTRL-S in what is commonly known as X-on/X-off protocol. Receipt of a CTRL-S input character will suspend output to that channel. Subsequent receipt of a CTRL-Q will resume the output. Also, when the VxWorks input buffer is almost full, a CTRL-S will be output to signal the other side to suspend transmission. When the input buffer is almost empty, a CTRL-Q will be output to signal the other side to resume transmission.

##### OPT\_7\_BIT

Strips the most significant bit from all bytes input from the device.

##### OPT\_MON\_TRAP

Enables the special monitor trap character, by default CTRL-X. When this character is received and this option is enabled, VxWorks will trap to the ROM resident monitor program. Note that this is quite drastic. All normal VxWorks functioning is suspended, and the computer system is entirely controlled by the monitor. Depending on the particular monitor, it may or may not be possible to restart VxWorks from the point of interruption. The default monitor trap character can be changed by calling **tyMonitorTrapSet()**.

**OPT\_ABORT**

Enables the special shell abort character, by default CTRL-C. When this character is received and this option is enabled, the VxWorks shell is restarted. This is useful for freeing a shell stuck in an unfriendly routine, such as one caught in an infinite loop or one that has taken an unavailable semaphore. For more information, see the *VxWorks Programmer's Guide: Shell*.

**OPT\_TERMINAL**

This is not a separate option bit. It is the value of the option word with all the above bits set.

**OPT\_RAW**

This is not a separate option bit. It is the value of the option word with none of the above bits set.

**I/O CONTROL FUNCTIONS**

The *tty* devices respond to the following `ioctl()` functions. The functions are defined in the header `ioLib.h`.

**FIOGETNAME**

Gets the file name of the file descriptor and copies it to the buffer referenced to by `nameBuf`:

```
status = ioctl (fd, FIOGETNAME, &nameBuf);
```

This function is common to all file descriptors for all devices.

**FIOSETOPTIONS, FIOOPTIONS**

Sets the device option word to the specified argument. For example, the call:

```
status = ioctl (fd, FIOOPTIONS, OPT_TERMINAL);
```

```
status = ioctl (fd, FIOSETOPTIONS, OPT_TERMINAL);
```

enables all the *tty* options described above, putting the device in a "normal" terminal mode. If the line protocol (`OPT_LINE`) is changed, the input buffer is flushed. The various options are described in `ioLib.h`.

**FIOGETOPTIONS**

Returns the current device option word:

```
options = ioctl (fd, FIOGETOPTIONS, 0);
```

**FIONREAD**

Copies to `nBytesUnread` the number of bytes available to be read in the device's input buffer:

```
status = ioctl (fd, FIONREAD, &nBytesUnread);
```

In line mode (`OPT_LINE` set), the `FIONREAD` function actually returns the number of characters available plus the number of lines in the buffer. Thus, if five lines of just `NEWLINES` were in the input buffer, it would return the value 10 (5 characters + 5 lines).

#### FIONWRITE

Copies to *nBytes* the number of bytes queued to be output in the device's output buffer:

```
status = ioctl (fd, FIONWRITE, &nBytes);
```

#### FIOFLUSH

Discards all the bytes currently in both the input and the output buffers:

```
status = ioctl (fd, FIOFLUSH, 0);
```

#### FIOWFLUSH

Discards all the bytes currently in the output buffer:

```
status = ioctl (fd, FIOWFLUSH, 0);
```

#### FIORFLUSH

Discards all the bytes currently in the input buffers:

```
status = ioctl (fd, FIORFLUSH, 0);
```

#### FIOCANCEL

Cancels a read or write. A task blocked on a read or write may be released by a second task using this `ioctl()` call. For example, a task doing a read can set a watchdog timer before attempting the read; the auxiliary task would wait on a semaphore. The watchdog routine can give the semaphore to the auxiliary task, which would then use the following call on the appropriate file descriptor:

```
status = ioctl (fd, FIOCANCEL, 0);
```

#### FIOBAUDRATE

Sets the baud rate of the device to the specified argument. For example, the call:

```
status = ioctl (fd, FIOBAUDRATE, 9600);
```

Sets the device to operate at 9600 baud. This request has no meaning on a pseudo terminal.

#### FIOISATTY

Returns `TRUE` for a *tty* device:

```
status = ioctl (fd, FIOISATTY, 0);
```

#### FIOPROTOHOOK

Adds a protocol hook function to be called for each input character. *pfunction* is a pointer to the protocol hook routine which takes two arguments of type *int* and returns values of type `STATUS` (`TRUE` or `FALSE`). The first argument passed is set by the user via the `FIOPROTOARG` function. The second argument is the input character. If no further processing of the character is required by the calling routine (the input routine of the driver), the protocol hook routine *pFunction* should return `TRUE`. Otherwise, it should return `FALSE`:

```
status = ioctl (fd, FIOPROTOHOOK, pFunction);
```

#### **FIOPROTOARG**

Sets the first argument to be passed to the protocol hook routine set by **FIOPROTOHOOK** function:

```
status = ioctl (fd, FIOPROTOARG, arg);
```

#### **FIORBUFSET**

Changes the size of the receive-side buffer to *size*:

```
status = ioctl (fd, FIORBUFSET, size);
```

#### **FIOWBUFSET**

Changes the size of the send-side buffer to *size*:

```
status = ioctl (fd, FIOWBUFSET, size);
```

Any other **ioctl()** request will return an error and set the status to **S\_ioLib\_UNKNOWN\_REQUEST**.

**INCLUDE FILES**    **tyLib.h, ioLib.h**

**SEE ALSO**        **ioLib, iosLib, tyCoDrv**, *VxWorks Programmer's Guide: I/O System*

## udpShow

<b>NAME</b>	<b>udpShow</b> – UDP information display routines
<b>ROUTINES</b>	<b>udpShowInit()</b> - initialize UDP show routines <b>udpstatShow()</b> - display statistics for the UDP protocol
<b>DESCRIPTION</b>	<p>This library provides routines to show UDP related statistics.</p> <p>Interpreting these statistics requires detailed knowledge of Internet network protocols. Information on these protocols can be found in the following books:</p> <p><i>TCP/IP Illustrated Volume II, The Implementation</i>, by Richard Stevens</p> <p><i>The Design and Implementation of the 4.4 BSD UNIX Operating System</i>, by Leffler, McKusick, Karels and Quarterman</p> <p>The <b>udpShowInit()</b> routine links the UDP show facility into the VxWorks system. This is performed automatically if <b>INCLUDE_NET_SHOW</b> and <b>INCLUDE_UDP</b> are defined.</p>
<b>SEE ALSO</b>	<b>netLib</b> , <b>netShow</b>

---

## unixDrv

<b>NAME</b>	<b>unixDrv</b> – UNIX-file disk driver (VxSim for Solaris and VxSim for HP)
<b>ROUTINES</b>	<b>unixDrv()</b> - install UNIX disk driver <b>unixDiskDevCreate()</b> - create a UNIX disk device <b>unixDiskInit()</b> - initialize a dosFs disk on top of UNIX
<b>DESCRIPTION</b>	<p>This driver emulates a VxWorks disk driver, but actually uses the UNIX file system to store the data. The VxWorks disk appears under UNIX as a single file. The UNIX file name, and the size of the disk, may be specified during the <b>unixDiskDevCreate()</b> call.</p>
<b>USER-CALLABLE ROUTINES</b>	<p>Most of the routines in this driver are accessible only through the I/O system. The routine <b>unixDrv()</b> must be called to initialize the driver and the <b>unixDiskDevCreate()</b> routine is used to create devices.</p>
<b>CREATING UNIX DISKS</b>	<p>Before a UNIX disk can be used, it must be created. This is done with the</p>

`unixDiskDevCreate()` call. The format of this call is:

```
BLK_DEV *unixDiskDevCreate
(
    char    *unixFile,      /* name of the UNIX file to use    */
    int     bytesPerBlk,    /* number of bytes per block       */
    int     blksPerTrack,   /* number of blocks per track      */
    int     nBlocks        /* number of blocks on this device  */
)
```

The UNIX file must be pre-allocated separately. This can be done using the UNIX `mkfile(8)` command. Note that you have to create an appropriately sized file. For example, to create a UNIX file system that is used as a common floppy dosFs file system, you would issue the command:

```
mkfile 1440k /tmp/floppy.dos
```

This will create space for a 1.44 Meg DOS floppy (1474560 bytes, or 2880 512-byte blocks).

The *bytesPerBlk* parameter specifies the size of each logical block on the disk. If *bytesPerBlk* is zero, 512 is the default.

The *blksPerTrack* parameter specifies the number of blocks on each logical track of the UNIX disk. If *blksPerTrack* is zero, the count of blocks per track will be set to *nBlocks* (i.e., the disk will be defined as having only one track). UNIX disk devices typically are specified with only one track.

The *nBlocks* parameter specifies the size of the disk, in blocks. If *nBlocks* is zero the size of the UNIX file specified, divided by the number of bytes per block, is used.

The formatting parameters (*bytesPerBlk*, *blksPerTrack*, and *nBlocks*) are critical only if the UNIX disk already contains the contents of a disk created elsewhere. In that case, the formatting parameters must be identical to those used when the image was created. Otherwise, they may be any convenient number.

Once the device has been created it still does not have a name or file system associated with it. This must be done by using the file system's device initialization routine (e.g., `dosFsDevInit()`). The dosFs and rt11Fs file systems also provide make-file-system routines (`dosFsMkfs()` and `rt11FsMkfs()`), which may be used to associate a name and file system with the block device and initialize that file system on the device using default configuration parameters.

The `unixDiskDevCreate()` call returns a pointer to a block device structure (`BLK_DEV`). This structure contains fields that describe the physical properties of a disk device and specify the addresses of routines within the UNIX disk driver. The `BLK_DEV` structure address must be passed to the desired file system (dosFs, rt11Fs, or rawFs) during the file system's device initialization or make-file-system routine. Only then is a name and file system associated with the device, making it available for use.

**unldLib**

As an example, to create a 200KB disk, 512-byte blocks, and only one track, the proper call would be:

```
BLK_DEV *pBlkDev;
pBlkDev = unixDiskDevCreate ("/tmp/filesys1", 512, 400, 400, 0);
```

This will attach the UNIX file `/tmp/filesys1` as a block device.

A convenience routine, `unixDiskInit()`, is provided to do the `unixDiskDevCreate()` followed by either a `dosFsMkFs()` or `dosFsDevInit()`, whichever is appropriate.

The format of this call is:

```
BLK_DEV *unixDiskInit
(
    char * unixFile, /* name of the UNIX file to use */
    char * volName,  /* name of the dosFs volume to use */
    int   nBytes     /* number of bytes in dosFs volume */
)
```

This call will create the UNIX disk if required.

**IOCTL** Only the `FIODISKFORMAT` request is supported; all other ioctl requests return an error, and set the task's `errno` to `S_ioLib_UNKNOWN_REQUEST`.

**SEE ALSO** `dosFsDevInit()`, `dosFsMkfs()`, `rt11FsDevInit()`, `rt11FsMkfs()`, `rawFsDevInit()`, *VxWorks Programmer's Guide: I/O System, Local File Systems*

---

## unldLib

**NAME** `unldLib` – object module unloading library

**ROUTINES** `unld()` - unload an object module by specifying a file name or module ID  
`unldByModuleId()` - unload an object module by specifying a module ID  
`unldByNameAndPath()` - unload an object module by specifying a name and path  
`unldByGroup()` - unload an object module by specifying a group number  
`reld()` - reload an object module

**DESCRIPTION** This library provides a facility for unloading object modules. Once an object module has been loaded into the system (using the facilities provided by `loadLib`), it can be removed from the system by calling one of the `unld...()` routines in this library.

Unloading of an object module does the following:

- (1) It frees the space allocated for text, data, and BSS segments, unless `loadModuleAt()` was called with specific addresses, in which case the user is responsible for freeing the space.

- (2) It removes all symbols associated with the object module from the system symbol table.
- (3) It removes the module descriptor from the module list.

Once the module is unloaded, any calls to routines in that module from other modules will fail unpredictably. The user is responsible for ensuring that no modules are unloaded that are used by other modules. **unld()** checks the hooks created by the following routines to ensure none of the unloaded code is in use by a hook:

```
taskCreateHookAdd()  
taskDeleteHookAdd()  
taskHookAdd()  
taskSwapHookAdd()  
taskSwitchHookAdd()
```

However, **unld()** *does not* check the hooks created by these routines:

```
etherInputHookAdd()  
etherOutputHookAdd()  
excHookAdd()  
rebootHookAdd()  
moduleCreateHookAdd()
```

The routines **unld()** and **reld()** are **shell commands**. That is, they are designed to be used only in the shell, and not in code running on the target. In future releases, calling **unld()** and **reld()** directly from code may not be supported.

**INCLUDE FILES**     **unldLib.h, moduleLib.h**

**SEE ALSO**         **loadLib, moduleLib**, *Tornado User's Guide: Cross-Development*

---

## usrAta

**NAME**             **usrAta** – ATA/ATAPI initialization

**ROUTINES**        **usrAtaConfig()** - mount a DOS file system from an ATA hard disk or a CDROM  
**usrAtaInit()** - initialize the hard disk driver

## usrConfig

<b>NAME</b>	<b>usrConfig</b> – user-defined system configuration library
<b>ROUTINES</b>	<b>usrInit()</b> - user-defined system initialization routine <b>usrRoot()</b> - the root task <b>usrClock()</b> - user-defined system clock interrupt routine
<b>DESCRIPTION</b>	<p>This library is the WRS-supplied configuration module for VxWorks. It contains the root task, the primary system initialization routine, the network initialization routine, and the clock interrupt routine.</p> <p>The include file <b>config.h</b> includes a number of system-dependent parameters used in this file.</p> <p>In an effort to simplify the presentation of the configuration of VxWorks, this file has been split into smaller files. These additional configuration source files are located in <code>../src/config/usrxxx.c</code> and are <b>#included</b> into this file below. This file contains the bulk of the code a customer is likely to customize.</p> <p>The module <b>usrDepend.c</b> contains checks that guard against unsupported configurations such as <b>INCLUDE_NFS</b> without <b>INCLUDE_RPC</b>. The module <b>usrKernel.c</b> contains the core initialization of the kernel which is rarely customized, but provided for information. The module <b>usrNetwork.c</b> now contains all network initialization code. Finally, the module <b>usrExtra.c</b> contains the conditional inclusion of the optional packages selected in <b>configAll.h</b>.</p> <p>The source code necessary for the configuration selected is entirely included in this file during compilation as part of a standard build in the board support package. No other make is necessary.</p>
<b>INCLUDE FILES</b>	<b>config.h</b>
<b>SEE ALSO</b>	<i>Tornado User's Guide: Getting Started, Cross-Development</i>

---

## usrFd

<b>NAME</b>	<b>usrFd</b> – floppy disk initialization
<b>ROUTINES</b>	<b>usrFdConfig()</b> - mount a DOS file system from a floppy disk

## usrFdiskPartLib

**NAME** usrFdiskPartLib – FDISK-style partition handler

**ROUTINES** `usrFdiskPartRead()` - read an FDISK-style partition table  
`usrFdiskPartCreate()` - create an FDISK-like partition table on a disk  
`usrFdiskPartShow()` - parse and display partition data

**DESCRIPTION** This module is provided is source code to accommodate various customizations of partition table handling, resulting from variations in the partition table format in a particular configuration. It is intended for use with `dpartCbio` partition manager.

This code supports both mounting MSDOS file systems and displaying partition tables written by MSDOS `FDISK.exe` or by any other MSDOS `FDISK.exe` compatible partitioning software.

The first partition table is contained within a hard drives Master Boot Record (MBR) sector, which is defined as sector one, cylinder zero, head zero or logical block address zero.

The mounting and displaying routines within this code will first parse the MBR partition tables entries (defined below) and also recursively parse any “extended” partition tables, which may reside within another sector further into the hard disk. MSDOS file systems within extended partitions are known to those familiar with the MSDOS `FDISK.exe` utility as “Logical drives within the extended partition”.

Here is a picture showing the layout of a single disk containing multiple MSDOS file systems:

```

+-----+
|<-----The entire disk----->|
|M                                     |
|B<---C:--->                           |
|R      /---- First extended partition-----\ |
|      E<---D:---><--Rest of the ext part----->|
|P      x                               |
|A      t      E<---E:--->E<Rest of the ext part->|
|R      x      x                               |
|T      t      t<-----F:----->|
+-----+
(Ext == extended partition sector)
C: is a primary partiion
D:, E:, and F: are logical drives within the extended partition.

```

A MS-DOS partition table resides within one sector on a hard disk. There is always one in the first sector of a hard disk partitioned with `FDISK.exe`. There first partition table may contain references to “extended” partition tables residing on other sectors if there are

multiple partitions. The first sector of the disk is the starting point. Partition tables are of the format:

Offset from the beginning of the sector	Description
0x1be	Partition 1 table entry (16 bytes)
0x1ce	Partition 2 table entry (16 bytes)
0x1de	Partition 3 table entry (16 bytes)
0x1ee	Partition 4 table entry (16 bytes)
0x1fe	Signature (0x55aa, 2 bytes)

Individual MSDOS partition table entries are of the format:

Offset	Size	Description
0x0	8 bits	boot type
0x1	8 bits	beginning sector head value
0x2	8 bits	beginning sector (2 high bits of cylinder#)
0x3	8 bits	beginning cylinder# (low order bits of cylinder#)
0x4	8 bits	system indicator
0x5	8 bits	ending sector head value
0x6	8 bits	ending sector (2 high bits of cylinder#)
0x7	8 bits	ending cylinder# (low order bits of cylinder#)
0x8	32 bits	number of sectors preceding the partition
0xc	32 bits	number of sectors in the partition

The Cylinder, Head and Sector values herein are not used, instead the 32-bit partition offset and size (also known as LBA addresses) are used exclusively to determine partition geometry.

If a non-partitioned disk is detected, in which case the 0'th block is a DosFs boot block rather than an MBR, the entire disk will be configured as partition 0, so that disks formatted with VxWorks and disks formatted on MS-DOS or Windows can be accepted interchangeably.

The **usrFdiskPartCreate()** will create a partition table with up to four partitions, which can be later used with **usrFdiskPartRead()** and **dpartCbio** to manage a partitioned disk on VxWorks.

However, it can not be guaranteed that this partition table can be used on another system due to several BIOS specific parameters in the boot area. If interchangeability via removable disks is a requirement, partition tables should be created and volumes should be formatted on the other system with which the data is to be interchanged.

---

**WARNING:** The partition decode function is recursive, up to the maximum number of partitions expected, which is no more than 24.

---

Sufficient stack space needs to be provided via **taskSpawn()** to accommodate the recursion level.

**SEE ALSO**      **dpartCbio**

---

## usrFsLib

**NAME**            **usrFsLib** – file system user interface subroutine library

**ROUTINES**        **cd()** - change the default directory  
**pwd()** - print the current default directory  
**mkdir()** - make a directory  
**rmdir()** - remove a directory  
**rm()** - remove a file  
**copyStreams()** - copy from/to specified streams  
**copy()** - copy *in* (or *stdin*) to *out* (or *stdout*)  
**chkdsk()** - perform consistency checking on a MS-DOS file system  
**dirList()** - list contents of a directory (multi-purpose)  
**ls()** - generate a brief listing of a directory  
**ll()** - generate a long listing of directory contents  
**lsr()** - list the contents of a directory and any of its subdirectories  
**llr()** - do a long listing of directory and all its subdirectories contents  
**cp()** - copy file into other file/directory.  
**mv()** - mv file into other directory.  
**xcopy()** - copy a hierarchy of files with wildcards  
**xdelete()** - delete a hierarchy of files with wildcards  
**attrib()** - modify MS-DOS file attributes on a file or directory  
**xattrib()** - modify MS-DOS file attributes of many files  
**diskFormat()** - format a disk  
**diskInit()** - initialize a file system on a block device  
**ioHelp()** - print a synopsis of I/O utility functions

**DESCRIPTION**    This library provides user-level utilities for managing file systems. These utilities may be used from Tornado Shell, the Target Shell or from an application.

### USAGE FROM TORNADO

Some of the functions in this library have counterparts of the same names built into the Tornado Shell (aka **Windsh**). The built-in functions perform similar functions on the Tornado host computer's I/O systems. Hence if one of such functions needs to be executed in order to perform any operation on the Target's I/O system, it must be preceded with an @ sign, e.g.: `ce > @ls "/sd0"` ce will list the directory of a disk named /sd0 on the target, while



## **usrIde**

-> `ls "/tmp"`

will list the contents of the `/tmp` directory on the host.

The target I/O system and the Tornado Shell running on the host, each have their own notion of current directory, which are not related, hence

-> `pwd`

will display the Tornado Shell current directory on the host file system, while

-> `@pwd`

will display the target's current directory on the target's console.

### **WILDCARDS**

Some of the functions herein support wildcard characters in argument strings where file or directory names are expected. The wildcards are limited to `"**"` which matches zero or more characters and `"?"` which matches any single characters. Files or directories with names beginning with a `"."` are not normally matched with the `"**"` wildcard.

### **DIRECTORY LISTING**

Directory listing is implemented in one function `dirList()`, which can be accessed using one of these four front-end functions:

`ls()`

produces a short list of files

`lsr()`

is like `ls()` but ascends into subdirectories

`ll()`

produces a detailed list of files, with file size, modification date attributes *etc.*

`llr()`

is like `ll()` but also ascends into subdirectories

All of the directory listing functions accept a name of a directory or a single file to list, or a name which contain wildcards, which will result in listing of all objects that match the wildcard string provided.

### **SEE ALSO**

`ioLib`, `dosFsLib`, `netDrv`, `nfsLib`, *VxWorks Programmer's Guide: Target Shell VxWorks Programmer's Guide: Tornado Users's Guide*

---

## **usrIde**

### **NAME**

`usrIde` – IDE initialization

### **ROUTINES**

`usrIdeConfig()` - mount a DOS file system from an IDE hard disk

---

## usrLib

<b>NAME</b>	<b>usrLib</b> – user interface subroutine library
<b>ROUTINES</b>	<p><b>help()</b> - print a synopsis of selected routines <b>netHelp()</b> - print a synopsis of network routines <b>bootChange()</b> - change the boot line <b>periodRun()</b> - call a function periodically <b>period()</b> - spawn a task to call a function periodically <b>repeatRun()</b> - call a function repeatedly <b>repeat()</b> - spawn a task to call a function repeatedly <b>sp()</b> - spawn a task with default parameters <b>checkStack()</b> - print a summary of each task's stack usage <b>i()</b> - print a summary of each task's TCB <b>ti()</b> - print complete information from a task's TCB <b>show()</b> - print information on a specified object <b>ts()</b> - suspend a task <b>tr()</b> - resume a task <b>td()</b> - delete a task <b>version()</b> - print VxWorks version information <b>m()</b> - modify memory <b>d()</b> - display memory <b>ld()</b> - load an object module into memory <b>devs()</b> - list all system-known devices <b>lkup()</b> - list symbols <b>lkAddr()</b> - list symbols whose values are near a specified value <b>mRegs()</b> - modify registers <b>pc()</b> - return the contents of the program counter <b>printErrno()</b> - print the definition of a specified error status value <b>printLogo()</b> - print the VxWorks logo <b>logout()</b> - log out of the VxWorks system <b>h()</b> - display or set the size of shell history <b>spyReport()</b> - display task activity data <b>spyTask()</b> - run periodic task activity reports <b>spy()</b> - begin periodic task activity reports <b>spyClkStart()</b> - start collecting task activity data <b>spyClkStop()</b> - stop collecting task activity data <b>spyStop()</b> - stop spying and reporting <b>spyHelp()</b> - display task monitoring help menu</p>
<b>DESCRIPTION</b>	This library consists of routines meant to be executed from the VxWorks shell. It provides useful utilities for task monitoring and execution, system information, symbol table management, <i>etc.</i>

## **usrScsi**

Many of the routines here are simply command-oriented interfaces to more general routines contained elsewhere in VxWorks. Users should feel free to modify or extend this library, and may find it preferable to customize capabilities by creating a new private library, using this one as a model, and appropriately linking the new one into the system.

Some routines here have optional parameters. If those parameters are zero, which is what the shell supplies if no argument is typed, default values are typically assumed.

A number of the routines in this module take an optional task name or ID as an argument. If this argument is omitted or zero, the “current” task is used. The current task (or “default” task) is the last task referenced. The **usrLib** library uses **taskIdDefault()** to set and get the last-referenced task ID, as do many other VxWorks routines.

**INCLUDE FILES**     **usrLib.h**

**SEE ALSO**         **usrFsLib**, **tarLib**, **spyLib**, *VxWorks Programmer's Guide: Target Shell*, **windsh**, *Tornado User's Guide: Shell*

---

## **usrScsi**

**NAME**             **usrScsi** – SCSI initialization

**ROUTINES**        **usrScsiConfig()** - configure SCSI peripherals

---

## vmBaseLib

<b>NAME</b>	<b>vmBaseLib</b> – base virtual memory support library
<b>ROUTINES</b>	<b>vmBaseLibInit()</b> - initialize base virtual memory support <b>vmBaseGlobalMapInit()</b> - initialize global mapping <b>vmBaseStateSet()</b> - change the state of a block of virtual memory <b>vmBasePageSizeGet()</b> - return the page size
<b>DESCRIPTION</b>	<p>This library provides the minimal MMU (Memory Management Unit) support needed in a system. Its primary purpose is to create cache-safe buffers for <b>cacheLib</b>. Buffers are provided to optimize I/O throughput.</p> <p>A call to <b>vmBaseLibInit()</b> initializes this library, thus permitting <b>vmBaseGlobalMapInit()</b> to initialize the MMU and set up MMU translation tables. Additionally, <b>vmBaseStateSet()</b> can be called to change the translation tables dynamically.</p> <p>This library is a release-bundled complement to <b>vmLib</b> and <b>vmShow</b>, modules that offer full-featured MMU support and virtual memory information display routines. The <b>vmLib</b> and <b>vmShow</b> libraries are distributed as the unbundled virtual memory support option, VxVMI.</p>
<b>CONFIGURATION</b>	Bundled MMU support is included in VxWorks when the configuration macro <b>INCLUDE_MMU_BASIC</b> is defined. If the configuration macro <b>INCLUDE_MMU_FULL</b> is also defined, the default is full MMU support (unbundled).
<b>INCLUDE FILES</b>	<b>sysLib.h</b> , <b>vmLib.h</b>
<b>SEE ALSO</b>	<b>vmLib</b> , <b>vmShow</b> , <i>VxWorks Programmer's Guide: Virtual Memory</i>

---

## vmLib

<b>NAME</b>	<b>vmLib</b> – architecture-independent virtual memory support library (VxVMI Opt.)
<b>ROUTINES</b>	<b>vmLibInit()</b> - initialize the virtual memory support module (VxVMI Opt.) <b>vmGlobalMapInit()</b> - initialize global mapping (VxVMI Opt.) <b>vmContextCreate()</b> - create a new virtual memory context (VxVMI Opt.) <b>vmContextDelete()</b> - delete a virtual memory context (VxVMI Opt.) <b>vmStateSet()</b> - change the state of a block of virtual memory (VxVMI Opt.) <b>vmStateGet()</b> - get the state of a page of virtual memory (VxVMI Opt.)

**vmLib**

**vmMap()** - map physical space into virtual space (VxVMI Opt.)  
**vmGlobalMap()** - map physical pages to virtual space in shared global virtual memory (VxVMI Opt.)  
**vmGlobalInfoGet()** - get global virtual memory information (VxVMI Opt.)  
**vmPageBlockSizeGet()** - get the architecture-dependent page block size (VxVMI Opt.)  
**vmTranslate()** - translate a virtual address to a physical address (VxVMI Opt.)  
**vmPageSizeGet()** - return the page size (VxVMI Opt.)  
**vmCurrentGet()** - get the current virtual memory context (VxVMI Opt.)  
**vmCurrentSet()** - set the current virtual memory context (VxVMI Opt.)  
**vmEnable()** - enable or disable virtual memory (VxVMI Opt.)  
**vmTextProtect()** - write-protect a text segment (VxVMI Opt.)

**DESCRIPTION**

This library provides an architecture-independent interface to the CPU's memory management unit (MMU). Although **vmLib** is implemented with architecture-specific libraries, application code need never reference directly the architecture-dependent code in these libraries.

A fundamental goal in the design of **vmLib** was to permit transparent backward compatibility with previous versions of VxWorks that did not use the MMU. System designers may opt to disable the MMU because of timing constraints, and some architectures do not support MMUs; therefore VxWorks functionality must not be dependent on the MMU. The resulting design permits a transparent configuration with no change in the programming environment (but the addition of several protection features, such as text segment protection) and the ability to disable virtual memory in systems that require it.

The **vmLib** library provides a mechanism for creating virtual memory contexts, **vmContextCreate()**. These contexts are not automatically created for individual tasks, but may be created dynamically by tasks, and swapped in and out in an application specific manner.

All virtual memory contexts share a global transparent mapping of virtual to physical memory for all of local memory and the local hardware device space (defined in **sysLib.c** for each board port in the **sysPhysMemDesc** data structure). When the system is initialized, all of local physical memory is accessible at the same address in virtual memory (this is done with calls to **vmGlobalMap()**). Modifications made to this global mapping in one virtual memory context appear in all virtual memory contexts. For example, if the exception vector table (which resides at address 0 in physical memory) is made read only by calling **vmStateSet()** on virtual address 0, the vector table will be read only in all virtual memory contexts.

Private virtual memory can also be created. When physical pages are mapped to virtual memory that is not in the global transparent region, this memory becomes accessible only in the context in which it was mapped. (The physical pages will also be accessible in the transparent translation at the physical address, unless the virtual pages in the global transparent translation region are explicitly invalidated.) State changes (writability, validity, etc.) to a section of private virtual memory in a virtual memory context do not

appear in other contexts. To facilitate the allocation of regions of virtual space, **vmGlobalInfoGet()** returns a pointer to an array of booleans describing which portions of the virtual address space are devoted to global memory. Each successive array element corresponds to contiguous regions of virtual memory the size of which is architecture-dependent and which may be obtained with a call to **vmPageBlockSizeGet()**. If the boolean array element is true, the corresponding region of virtual memory, a “page block”, is reserved for global virtual memory and should not be used for private virtual memory. (If **vmMap()** is called to map virtual memory previously defined as global, the routine will return an error.)

All the state information for a block of virtual memory can be set in a single call to **vmStateSet()**. It performs parameter checking and checks the validity of the specified virtual memory context. It may also be used to set architecture-dependent state information. See **vmLib.h** for additional architecture-dependent state information.

The routine **vmContextShow()** in **vmShow** displays the virtual memory context for a specified context. For more information, see the manual entry for this routine.

- CONFIGURATION** Full MMU support (**vmLib**, and optionally, **vmShow**) is included in VxWorks when the configuration macro **INCLUDE\_MMU\_FULL** is defined. If the configuration macro **INCLUDE\_MMU\_BASIC** is also defined, the default is full MMU support (unbundled).
- The **sysLib.c** library contains a data structure called **sysPhysMemDesc**, which is an array of **PHYS\_MEM\_DESC** structures. Each element of the array describes a contiguous section of physical memory. The description of this memory includes its physical address, the virtual address where it should be mapped (typically, this is the same as the physical address, but not necessarily so), an initial state for the memory, and a mask defining which state bits in the state value are to be set. Default configurations are defined for each board support package (BSP), but these mappings may be changed to suit user-specific system configurations. For example, the user may need to map additional VME space where the backplane network interface data structures appear.
- AVAILABILITY** This library and **vmShow** are distributed as the unbundled virtual memory support option, VxVMI. A scaled down version, **vmBaseLib**, is provided with VxWorks for systems that do not permit optional use of the MMU, or for architectures that require certain features of the MMU to perform optimally (in particular, architectures that rely heavily on caching, but do not support bus snooping, and thus require the ability to mark inter-processor communications buffers as non-cacheable.) Most routines in **vmBaseLib** are referenced internally by VxWorks; they are not callable by application code.
- INCLUDE FILES** **vmLib.h**
- SEE ALSO** **sysLib**, **vmShow**, *VxWorks Programmer's Guide: Virtual Memory*

## vmShow

<b>NAME</b>	<b>vmShow</b> – virtual memory show routines (VxVMI Opt.)
<b>ROUTINES</b>	<b>vmShowInit()</b> - include virtual memory show facility (VxVMI Opt.) <b>vmContextShow()</b> - display the translation table for a context (VxVMI Opt.)
<b>DESCRIPTION</b>	<p>This library contains virtual memory information display routines.</p> <p>The routine <b>vmShowInit()</b> links this facility into the VxWorks system. It is called automatically when this facility is configured into VxWorks using either of the following methods:</p> <p>If you use the configuration header files, define both <b>INCLUDE_MMU_FULL</b> and <b>INCLUDE_SHOW_ROUTINES</b> in <b>config.h</b>.</p> <p>If you use the Tornado project facility, select <b>INCLUDE_MMU_FULL_SHOW</b>.</p>
<b>AVAILABILITY</b>	This module and <b>vmLib</b> are distributed as the unbundled virtual memory support option, VxVMI.
<b>INCLUDE FILES</b>	<b>vmLib.h</b>
<b>SEE ALSO</b>	<b>vmLib</b> , <i>VxWorks Programmer's Guide: Virtual Memory</i>

---

## vxLib

<b>NAME</b>	<b>vxLib</b> – miscellaneous support routines
<b>ROUTINES</b>	<b>vxTas()</b> - C-callable atomic test-and-set primitive <b>vxMemArchProbe()</b> - architecture specific part of <b>vxMemProbe()</b> <b>vxMemProbe()</b> - probe an address for a bus error <b>vxSSEnable()</b> - enable the superscalar dispatch (MC68060) <b>vxSSDisable()</b> - disable the superscalar dispatch (MC68060) <b>vxPowerModeSet()</b> - set the power management mode (PowerPC, SH, x86) <b>vxPowerModeGet()</b> - get the power management mode (PowerPC, SH, x86) <b>vxPowerDown()</b> - place the processor in reduced-power mode (PowerPC, SH) <b>vxCr0Get()</b> - get a content of the Control Register 0 (x86) <b>vxCr0Set()</b> - set a value to the Control Register 0 (x86) <b>vxCr2Get()</b> - get a content of the Control Register 2 (x86) <b>vxCr2Set()</b> - set a value to the Control Register 2 (x86) <b>vxCr3Get()</b> - get a content of the Control Register 3 (x86)

**vxCr3Set()** - set a value to the Control Register 3 (x86)  
**vxCr4Get()** - get a content of the Control Register 4 (x86)  
**vxCr4Set()** - set a value to the Control Register 4 (x86)  
**vxEflagsGet()** - get a content of the EFLAGS register (x86)  
**vxEflagsSet()** - set a value to the EFLAGS register (x86)  
**vxDrGet()** - get a content of the Debug Register 0 to 7 (x86)  
**vxDrSet()** - set a value to the Debug Register 0 to 7 (x86)  
**vxTssGet()** - get a content of the TASK register (x86)  
**vxTssSet()** - set a value to the TASK register (x86)  
**vxGdtrGet()** - get a content of the Global Descriptor Table Register (x86)  
**vxIdtrGet()** - get a content of the Interrupt Descriptor Table Register (x86)  
**vxLdtrGet()** - get a content of the Local Descriptor Table Register (x86)

**DESCRIPTION** This module contains miscellaneous VxWorks support routines.

**INCLUDE FILES** vxLib.h

## wdbLib

<b>NAME</b>	<b>wdbLib</b> – WDB agent context management library
<b>ROUTINES</b>	<b>wdbSystemSuspend()</b> - suspend the system.
<b>DESCRIPTION</b>	This library provides a routine to transfer control from the run time system to the WDB agent running in external mode. This agent in external mode allows a system-wide control, including ISR debugging, from a host tool (e.g.: <b>Crosswind</b> , <b>WindSh</b> ...) through the target server and the WDB communication link.
<b>INCLUDE FILES</b>	<b>wdb/wdbLib.h</b>
<b>SEE ALSO</b>	<i>API Guide: WTX Protocol, Tornado User's Guide: Overview</i>

---

## wdbUserEvtLib

<b>NAME</b>	<b>wdbUserEvtLib</b> – WDB user event library
<b>ROUTINES</b>	<b>wdbUserEvtLibInit()</b> - include the WDB user event library <b>wdbUserEvtPost()</b> - post a user event string to host tools.
<b>DESCRIPTION</b>	This library contains routines for sending WDB User Events. The event is sent through the WDB agent, the WDB communication link and the target server to the host tools that have registered for it. The event received by host tools will be a WTX user event string.
<b>INCLUDE FILES</b>	<b>wdb/wdbLib.h</b>
<b>SEE ALSO</b>	<i>API Guide: WTX Protocol</i>

---

## wdLib

<b>NAME</b>	<b>wdLib</b> – watchdog timer library
<b>ROUTINES</b>	<b>wdCreate()</b> - create a watchdog timer <b>wdDelete()</b> - delete a watchdog timer <b>wdStart()</b> - start a watchdog timer <b>wdCancel()</b> - cancel a currently counting watchdog
<b>DESCRIPTION</b>	<p>This library provides a general watchdog timer facility. Any task may create a watchdog timer and use it to run a specified routine in the context of the system-clock ISR, after a specified delay.</p> <p>Once a timer has been created with <b>wdCreate()</b>, it can be started with <b>wdStart()</b>. The <b>wdStart()</b> routine specifies what routine to run, a parameter for that routine, and the amount of time (in ticks) before the routine is to be called. (The timeout value is in ticks as determined by the system clock; see <b>sysClkRateSet()</b> for more information.) After the specified delay ticks have elapsed (unless <b>wdCancel()</b> is called first to cancel the timer) the timeout routine is invoked with the parameter specified in the <b>wdStart()</b> call. The timeout routine is invoked whether the task which started the watchdog is running, suspended, or deleted.</p> <p>The timeout routine executes only once per <b>wdStart()</b> invocation; there is no need to cancel a timer with <b>wdCancel()</b> after it has expired, or in the expiration callback itself.</p> <p>Note that the timeout routine is invoked at interrupt level, rather than in the context of the task. Thus, there are restrictions on what the routine may do. Watchdog routines are constrained to the same rules as interrupt service routines. For example, they may not take semaphores, issue other calls that may block, or use I/O system routines like <b>printf()</b>.</p>
<b>EXAMPLE</b>	<p>In the fragment below, if <b>maybeSlowRoutine()</b> takes more than 60 ticks, <b>logMsg()</b> will be called with the string as a parameter, causing the message to be printed on the console. Normally, of course, more significant corrective action would be taken.</p> <pre>WDG_ID wid = wdCreate (); wdStart (wid, 60, logMsg, "Help, I've timed out!"); maybeSlowRoutine ();          /* user-supplied routine */ wdCancel (wid);</pre>
<b>INCLUDE FILES</b>	<b>wdLib.h</b>
<b>SEE ALSO</b>	<b>logLib</b> , <i>VxWorks Programmer's Guide: Basic OS</i>

## wdShow

<b>NAME</b>	<b>wdShow</b> – watchdog show routines
<b>ROUTINES</b>	<b>wdShowInit()</b> - initialize the watchdog show facility <b>wdShow()</b> - show information about a watchdog
<b>DESCRIPTION</b>	<p>This library provides routines to show watchdog statistics, such as watchdog activity, a watchdog routine, <i>etc.</i></p> <p>The routine <b>wdShowInit()</b> links the watchdog show facility into the VxWorks system. It is called automatically when this show facility is configured into VxWorks using either of the following methods:</p> <ul style="list-style-type: none"><li>– If you use the configuration header files, define <b>INCLUDE_SHOW_ROUTINES</b> in <b>config.h</b>.</li><li>– If you use the Tornado project facility, select <b>INCLUDE_WATCHDOGS_SHOW</b>.</li></ul>
<b>INCLUDE FILES</b>	<b>wdLib.h</b>
<b>SEE ALSO</b>	<b>wdLib</b> , <i>VxWorks Programmer's Guide: Basic OS, Target Shell</i> , <b>windsh</b> , <i>Tornado User's Guide: Shell</i>

---

## wvFileUploadPathLib

<b>NAME</b>	<b>wvFileUploadPathLib</b> – file destination for event data
<b>ROUTINES</b>	<b>fileUploadPathLibInit()</b> - initialize the <b>wvFileUploadPathLib</b> library (Windview) <b>fileUploadPathCreate()</b> - create a file for depositing event data (Windview) <b>fileUploadPathClose()</b> - close the event-destination file (WindView) <b>fileUploadPathWrite()</b> - write to the event-destination file (WindView)
<b>DESCRIPTION</b>	<p>This file contains routines that write events to a file rather than uploading them to the host using a type of socket connection. If the file indicated is a TSFS file, this routine has the same result as uploading to a host file using other methods, allowing it to replace <b>evtRecv()</b>. The file can be created anywhere, however, and event data can be kept on the target if desired.</p>
<b>SEE ALSO</b>	<b>wvSockUploadPathLib</b> , <b>wvTsfsUploadPathLib</b>

---

## wvLib

<b>NAME</b>	<b>wvLib</b> – event logging control library (WindView)
<b>ROUTINES</b>	<b>wvLibInit()</b> - initialize <b>wvLib</b> - first step (WindView) <b>wvLibInit2()</b> - initialize <b>wvLib</b> - final step (WindView) <b>wvEvtLogInit()</b> - initialize an event log (WindView) <b>wvEvtLogStart()</b> - start logging events to the buffer (WindView) <b>wvEvtLogStop()</b> - stop logging events to the buffer (WindView) <b>wvEvtClassSet()</b> - set the class of events to log (WindView) <b>wvEvtClassGet()</b> - get the current set of classes being logged (WindView) <b>wvEvtClassClear()</b> - clear the specified class of events from those being logged (WindView) <b>wvEvtClassClearAll()</b> - clear all classes of events from those logged (WindView) <b>wvObjInstModeSet()</b> - set object instrumentation on/off (WindView) <b>wvObjInst()</b> - instrument objects (WindView) <b>wvSigInst()</b> - instrument signals (WindView) <b>wvEventInst()</b> - instrument VxWorks Events (WindView) <b>wvEvent()</b> - log a user-defined event (WindView) <b>wvUploadStart()</b> - start upload of events to the host (WindView) <b>wvUploadStop()</b> - stop upload of events to host (WindView) <b>wvUploadTaskConfig()</b> - set priority and stack size of <b>tWVUpload</b> task (WindView) <b>wvLogHeaderCreate()</b> - create the event-log header (WindView) <b>wvLogHeaderUpload()</b> - transfer the log header to the host (WindView) <b>wvEvtBufferGet()</b> - return the ID of the WindView event buffer (WindView) <b>wvTaskNamesPreserve()</b> - preserve an extra copy of task name events (WindView) <b>wvTaskNamesUpload()</b> - upload preserved task name events (WindView)

**DESCRIPTION** This library contains routines that control event collection and upload of event data from the target to various destinations. The routines define the interface for the target component of WindView. When event data has been collected, the routines in this library are used to produce event logs that can be understood by the WindView host tools.

An event log is made up of a header, followed by the task names of each task present in the system when the log is started, followed by a string of events produced by the various event points throughout the kernel and associated libraries. In general, this information is gathered and stored temporarily on the target, and later uploaded to the host in the proper order to form an event log. The routines in this file can be used to create logs in various ways, depending on which routines are called, and in which order the routines are called.

There are three methods for uploading event logs. The first is to defer upload of event data until after logging has been stopped in order to eliminate events associated with upload activity from the event log. The second is to continuously upload event data as it is gathered. This allows the collection of very large event logs, that may contain more events

than the target event buffer can store at one time. The third is to defer upload of the data until after a target reboot. This method allows event data to continuously overwrite earlier data in the event buffer, creating a log of the events leading to a target failure (a post-mortem event log).

Each of these three methods is explained in more detail in *CREATING AN EVENT LOG*.

#### EVENT BUFFERS AND UPLOAD PATHS

Many of the routines in **wvLib** require access to the buffer used to store event data (the event buffer) and to the communication paths from the target to the host (the upload paths). Both the buffer and the path are referenced with IDs that provide **wvLib** with the appropriate information for access.

The event buffering mechanism used by **wvLib** is provided by **rBuffLib**. The upload paths available for use with **wvLib** are provided by **wvFileUploadPathLib**, **wvTsfsUploadPathLib** and **wvSockUploadPathLib**.

The upload mechanism backs off and retries writing to the upload path if an error occurs during the write attempt with the **errno** **EAGAIN** or **EWOULDBLOCK**. Two global variables are used to set the amount of time to back off and the number of retries. The variables are:

```
int wvUploadMaxAttempts /* number of attempts to try writing */
int wvUploadRetryBackoff /* delay between tries (in ticks - 60/sec) */
```

#### INITIALIZATION

This library is initialized in two steps. The first step, done by calling **wvLibInit()**, associates event logging routines to system objects. This is done when the kernel is initialized. The second step, done by calling **wvLibInit2()**, associates all other event logging routines with the appropriate event points. Initialization is done automatically when **INCLUDE\_WINDVIEW** is defined.

Before event logging can be started, and each time a new event buffer is used to store logged events, **wvEvtLogInit()** must be called to bind the event logging routines to a specific buffer.

#### DETERMINING WHICH EVENTS ARE COLLECTED

There are three classes of events that can be collected. They are:

```
WV_CLASS_1 /* Events causing context switches */
WV_CLASS_2 /* Events causing task-state transitions */
WV_CLASS_3 /* Events from object and system libraries */
```

The second class includes all of the events contained within the first class, plus additional events causing task-state transitions but not causing context switches. The third class contains all of the second, and allows logging of events within system libraries. It can also be limited to specific objects or groups of objects:

- Using **wvObjInst()** allows individual objects (for example, **sem1**) to be instrumented.
- Using **wvSigInst()** allows signals to be instrumented.

- Using `wvObjInstModeSet()` allows finer control over what type of objects are instrumented. `wvObjInstModeSet()` allows types of system objects (for example, semaphores, watchdogs) to be instrumented as they are created.

Logging events in Class 3 generates the most data, which may be helpful during analysis of the log. It is also the most intrusive on the system, and may affect timing and performance. Class 2 is more intrusive than Class 1. In general, it is best to use the lowest class that still provides the required level of detail.

To manipulate the class of events being logged, the following routines can be used: `wvEvtClassSet()`, `wvEvtClassGet()`, `wvEvtClassClear()`, and `wvEvtClassClearAll()`. To log a user-defined event, `wvEvent()` can be used. It is also possible to log an event from any point during execution using `e()`, located in `dbgLib`.

#### CONTROLLING EVENT LOGGING

Once the class of events has been specified, event logging can be started with `wvEvtLogStart()` and stopped with `wvEvtLogStop()`.

#### CREATING AN EVENT LOG

An event log consists of a header, a section of task names, and a list of events logged after calling `wvEvtLogStart()`. As discussed above, there are three common ways to upload an event log.

##### Deferred Upload

When creating an event log by uploading the event data after event logging has been stopped (deferred upload), the following series of calls can be used to start and stop the collection. In this example the memory allocated to store the log header is in the system partition. The event buffer should be allocated from the system memory partition as well. Error checking has been eliminated to simplify the example.

```

/* wvLib and rBuffLib initialized at system start up */
#include "vxWorks.h"
#include "wvLib.h"
#include "private/wvBufferP.h"
#include "private/wvUploadPathP.h"
#include "private/wvFileUploadPathLibP.h"
BUFFER_ID      bufId;
UPLOAD_ID      pathId;
WV_UPLOAD_TASK_ID  upTaskId;
WV_LOG_HEADER_ID  hdrId;
/*
 * To prepare the event log and start logging:
 */
/* Create event buffer in memSysPart, yielding bufId. */
wvEvtLogInit (bufId);
hdrId = wvLogHeaderCreate (memSysPartId);
wvEvtClassSet (WV_CLASS_1);          /* set to log class 1 events */

```

```
wvEvtLogStart ();
/*
 * To stop logging and complete the event log.
 */
wvEvtLogStop ();
/* Create an upload path using wvFileUploadPathLib, yielding pathId. */
wvLogHeaderUpload (hdrId, pathId);
upTaskId = wvUploadStart (bufId, pathId, TRUE);
wvUploadStop (upTaskId);
/* Close the upload path and destroy the event buffer */
```

Routines which can be used as they are, or modified to meet the users needs, are located in `usrWindview.c`. These routines, `wvOn()` and `wvOff()`, provide a way to produce useful event logs without using the host user interface of WindView.

#### Continuous Upload

When uploading event data as it is still being logged to the event buffer (continuous upload), simply rearrange the above calls:

```
/* Includes and declarations. */
/*
 * To prepare the event log and start logging:
 */
/* Create event buffer in memSysPart, yielding bufId. */
/* Create an upload path, yielding pathId. */
wvEvtLogInit (bufId);
upTaskId = wvUploadStart (bufId, pathId, TRUE);
hdrId = wvLogHeaderCreate (memSysPartId);
wvLogHeaderUpload (hdrId, pathId);
wvEvtClassSet (WV_CLASS_1);          /* set to log class 1 events */
wvEvtLogStart ();
/*
 * To stop logging and complete the event log:
 */
wvEvtLogStop ();
wvUploadStop (upTaskId);
/* Close the upload path and destroy the event buffer */
```

#### Post-Mortem Event Collection

This library also contains routines that preserve task name information throughout event logging in order to produce post-mortem event logs: `wvTaskNamesPreserve()` and `wvTaskNamesUpload()`.

Post-mortem event logs typically contain events leading up to a target failure. The memory containing the information to be stored in the log must not be zeroed when the system reboots. The event buffer is set up to allow event data to be logged to it continuously, overwriting the data collected earlier. When event logging is stopped, either

by a system failure or at the request of the user, the event buffer may not contain the first events logged due to the overwriting. As tasks are created the `EVENT_TASKNAME` that is used by the WindView host tools to associate a task ID with a task name can be overwritten, while other events pertaining to that task ID may still be present in the event buffer. In order to assure that the WindView host tools can assign a task name to a context, a copy of all task name events can be preserved outside the event buffer and uploaded separately from the event buffer.

Note that several of the routines in `wvLib`, including `wvTaskNamesPreserve()`, take a memory partition ID as an argument. This allows memory to be allocated from a user-specified partition. For post-mortem data collection, the memory partition should be within memory that is not zeroed upon system reboot. The event buffer, preserved task names, and log header should be stored in this partition.

Generating a post-mortem event log is similar to generating a deferred upload log. Typically event logging is stopped due to a system failure, but it may be stopped in any way. To retrieve the log header, task name buffer, and event buffer after a target reboot, these IDs must be remembered or stored along with the collected information in the non-zeroed memory. Also, the event buffer should be set to allow continuous logging by overwriting earlier event data. The following produces a post-mortem log. The non-zeroed memory partition has the ID `postMortemPartId`.

```

/* Includes, as in the examples above. */
BUFFER_ID      bufId;
UPLOAD_ID      pathId;
WV_UPLOAD_TASK_ID  upTaskId;
WV_LOG_HEADER_ID  hdrId;
WV_TASKBUF_ID  taskBufId;
/*
 * To prepare the event log and start logging:
 */
/*
 * Create event buffer in non-zeroed memory, allowing overwrite,
 * @ yielding bufId.
 */
wvEvtLogInit (bufId);
taskBufId = wvTaskNamesPreserve (postMortemPartId, 32);
hdrId = wvLogHeaderCreate (postMortemPartId);
wvEvtClassSet (WV_CLASS_1);          /* set to log class 1 events */
wvEvtLogStart ();
/*
 * System fails and reboots. Note that taskBufId, bufId and
 * @ hdrId must be preserved through the reboot so they can be
 * @ used to upload the data.
 */
/* Create an upload path, yielding pathId. */
wvLogHeaderUpload (hdrId, pathId);

```

```

upTaskId = wvUploadStart (bufId, pathId, TRUE);
wvUploadStop (upTaskId);
wvTaskNamesUpload (taskBufId, pathId);
/* Close the upload path and destroy the event buffer */

```

**INCLUDE FILES** wvLib.h, eventP.h

**SEE ALSO** rBuffLib, wvFileUploadPathLib, wvSockUploadPathLib, wvTsfsUploadPathLib, *WindView User's Guide*

---

## wvNetLib

**NAME** wvNetLib – WindView for Networking Interface Library

**ROUTINES**

- wvNetEnable() - begin reporting network events to WindView
- wvNetDisable() - end reporting of network events to WindView
- wvNetLevelAdd() - enable network events with specific priority level
- wvNetLevelRemove() - disable network events with specific priority level
- wvNetEventEnable() - activate specific network events
- wvNetEventDisable() - deactivate specific network events
- wvNetAddressFilterSet() - specify an address filter for events
- wvNetAddressFilterClear() - remove the address filter for events
- wvNetPortFilterSet() - specify an address filter for events
- wvNetPortFilterClear() - remove the port number filter for events

**DESCRIPTION** This library provides the user interface to the network-related events for the WindView system visualization tool. These events are divided into two WindView classes. The `NET_CORE_EVENT` class indicates events directly related to data transfer. All other events (such as memory allocation and API routines) use the `NET_AUX_EVENT` class. Within each class, events are assigned one of eight priority levels. The four highest priority levels (**EMERGENCY**, **ALERT**, **CRITICAL**, and **ERROR**) indicate the occurrence of errors and the remaining four (**WARNING**, **NOTICE**, **INFO**, and **VERBOSE**) provide progressively more detailed information about the internal processing in the network stack.

**USER INTERFACE** If WindView support is included, the `wvNetStart()` and `wvNetStop()` routines will enable and disable event reporting for the network stack. The start routine takes a single parameter specifying the minimum priority level for all network components. That setting may be modified with the `wvNetLevelAdd()` and `wvNetLevelRemove()` routines. Individual events may be included or removed with the `wvNetEventEnable()` and `wvNetDisable()` routines.

The `wvNetAddressFilterSet()` and `wvNetPortFilterSet()` routines provide further screening for some events.

**SEE ALSO** *WindView for Tornado User's Guide*

---

## wvSockUploadPathLib

<b>NAME</b>	<b>wvSockUploadPathLib</b> – socket upload path library
<b>ROUTINES</b>	<b>sockUploadPathLibInit()</b> - initialize <b>wvSockUploadPathLib</b> library (Windview) <b>sockUploadPathCreate()</b> - establish an upload path to the host using a socket (Windview) <b>sockUploadPathClose()</b> - close the socket upload path (Windview) <b>sockUploadPathWrite()</b> - write to the socket upload path (Windview)
<b>DESCRIPTION</b>	This file contains routines that are used by <b>wvLib</b> to pass event data from the target buffers to the host. This particular event-upload path opens a normal network socket connected with the WindView host process to transfer the data.
<b>SEE ALSO</b>	<b>wvTsfsUploadPathLib</b> , <b>wvFileUploadPathLib</b>

---

## wvTmrLib

<b>NAME</b>	<b>wvTmrLib</b> – timer library (WindView)
<b>ROUTINES</b>	<b>wvTmrRegister()</b> - register a timestamp timer (WindView)
<b>DESCRIPTION</b>	This library allows a WindView timestamp timer to be registered. When this timer is enabled, events are tagged with a timestamp as they are logged.  Seven routines are required: a timestamp routine, a timestamp routine that guarantees interrupt lockout, a routine that enables the timer driver, a routine that disables the timer driver, a routine that specifies the routine to run when the timer hits a rollover, a routine that returns the period of the timer, and a routine that returns the frequency of the timer.
<b>SEE ALSO</b>	<b>wvLib</b> , <i>WindView User's Guide</i>

## wvTsfsUploadPathLib

<b>NAME</b>	<b>wvTsfsUploadPathLib</b> – target host connection library using TSFS
<b>ROUTINES</b>	<b>tsfsUploadPathLibInit()</b> - initialize <b>wvTsfsUploadPathLib</b> library (Windview) <b>tsfsUploadPathCreate()</b> - open an upload path to the host using a TSFS socket (Windview) <b>tsfsUploadPathClose()</b> - close the TSFS-socket upload path (Windview) <b>tsfsUploadPathWrite()</b> - write to the TSFS upload path (Windview)
<b>DESCRIPTION</b>	This library contains routines that are used by <b>wvLib</b> to transfer event data from the target to the host. This transfer mechanism uses the socket functionality of the Target Server File System (TSFS), and can therefore be used without including any socket or network facilities within the target.
<b>SEE ALSO</b>	<b>wvSockUploadPathLib</b> , <b>wvFileUploadPathLib</b>

---

## zbufLib

<b>NAME</b>	<b>zbufLib</b> – zbuf interface library
<b>ROUTINES</b>	<b>zbufCreate()</b> - create an empty zbuf <b>zbufDelete()</b> - delete a zbuf <b>zbufInsert()</b> - insert a zbuf into another zbuf <b>zbufInsertBuf()</b> - create a zbuf segment from a buffer and insert into a zbuf <b>zbufInsertCopy()</b> - copy buffer data into a zbuf <b>zbufExtractCopy()</b> - copy data from a zbuf to a buffer <b>zbufCut()</b> - delete bytes from a zbuf <b>zbufSplit()</b> - split a zbuf into two separate zbufs <b>zbufDup()</b> - duplicate a zbuf <b>zbufLength()</b> - determine the length in bytes of a zbuf <b>zbufSegFind()</b> - find the zbuf segment containing a specified byte location <b>zbufSegNext()</b> - get the next segment in a zbuf <b>zbufSegPrev()</b> - get the previous segment in a zbuf <b>zbufSegData()</b> - determine the location of data in a zbuf segment <b>zbufSegLength()</b> - determine the length of a zbuf segment
<b>DESCRIPTION</b>	<p>This library contains routines to create, build, manipulate, and delete zbufs. Zbufs, also known as “zero copy buffers,” are a data abstraction designed to allow software modules to share buffers without unnecessarily copying data.</p> <p>To support the data abstraction, the subroutines in this library hide the implementation details of zbufs. This also maintains the library’s independence from any particular implementation mechanism, thus permitting the zbuf interface to be used with other buffering schemes.</p> <p>Zbufs have three essential properties. First, a zbuf holds a sequence of bytes. Second, these bytes are organized into one or more segments of contiguous data, although the successive segments themselves are not usually contiguous. Third, the data within a segment may be shared with other segments; that is, the data may be in use by more than one zbuf at a time.</p>
<b>ZBUF TYPES</b>	<p>The following data types are used in managing zbufs:</p> <p><b>ZBUF_ID</b> An arbitrary (but unique) integer that identifies a particular zbuf.</p> <p><b>ZBUF_SEG</b> An arbitrary (but unique within a single zbuf) integer that identifies a segment within a zbuf.</p>

### ADDRESSING BYTES IN ZBUFS

The bytes in a zbuf are addressed by the combination *zbufSeg*, *offset*. The *offset* may be

positive or negative, and is simply the number of bytes from the beginning of the segment *zbufSeg*.

A *zbufSeg* can be specified as **NULL**, to identify the segment at the beginning of a zbuf. If *zbufseg* is **NULL**, *offset* is the absolute offset to any byte in the zbuf. However, it is more efficient to identify a zbuf byte location relative to the *zbufSeg* that contains it; see **zbufSegFind()** to convert any *zbufSeg*, *offset* pair to the most efficient equivalent.

Negative *offset* values always refer to bytes before the corresponding *zbufSeg*, and are not usually the most efficient address formulation (though using them may save your program other work in some cases).

The following special *offset* values, defined as constants, allow you to specify the very beginning or the very end of an entire zbuf, regardless of the *zbufSeg* value:

**ZBUF\_BEGIN**

The beginning of the entire zbuf.

**ZBUF\_END**

The end of the entire zbuf (useful for appending to a zbuf; see below).

**INSERTION AND LIMITS ON OFFSETS**

An *offset* is not valid if it points outside the zbuf. Thus, to address data currently within an N-byte zbuf, the valid offsets relative to the first segment are 0 through N-1.

Insertion routines are a special case: they obey the usual convention, but they use *offset* to specify where the new data begins after the insertion is complete. Therefore, the original zbuf data is always inserted just before the byte location addressed by the *offset* value. The value of this convention is that it permits inserting (or concatenating) data either before or after the existing data. To insert before all the data currently in a zbuf segment, use 0 as *offset*. To insert after all the data in an N-byte segment, use N as *offset*. An *offset* of N-1 inserts the data just before the last byte in an N-byte segment.

An *offset* of 0 is always a valid insertion point; for an empty zbuf, 0 is the only valid *offset* (and **NULL** the only valid *zbufSeg*).

**SHARING DATA**

The routines in this library avoid copying segment data whenever possible. Thus, by passing and manipulating **ZBUF\_IDs** rather than copying data, multiple programs can communicate with greater efficiency. However, each program must be aware of data sharing: changes to the data in a zbuf segment are visible to all zbuf segments that reference the data.

To alter your own program's view of zbuf data without affecting other programs, first use **zbufDup()** to make a new zbuf; then you can use an insertion or deletion routine, such as **zbufInsertBuf()**, to add a segment that only your program sees (until you pass a zbuf containing it to another program). It is safest to do all direct data manipulation in a private buffer, before enrolling it in a zbuf: in principle, you should regard all zbuf segment data as shared.

Once a data buffer is enrolled in a zbuf segment, the zbuf library is responsible for noticing when the buffer is no longer in use by any program, and freeing it. To support this, **zbufInsertBuf()** requires that you specify a callback to a free routine each time you build a zbuf segment around an existing buffer. You can use this callback to notify your application when a data buffer is no longer in use.

**VXWORKS AE PROTECTION DOMAINS**

Under VxWorks AE, this feature is restricted to the kernel protection domain. This restriction does not apply under non-AE versions of VxWorks.

To use this feature, include the following component: **INCLUDE\_ZBUF SOCK**

**SEE ALSO**      **zbufSockLib**

---

## **zbufSockLib**

**NAME**            **zbufSockLib** – zbuf socket interface library

**ROUTINES**        **zbufSockLibInit()** - initialize the zbuf socket interface library  
**zbufSockSend()** - send zbuf data to a TCP socket  
**zbufSockSendto()** - send a zbuf message to a UDP socket  
**zbufSockBufSend()** - create a zbuf from user data and send it to a TCP socket  
**zbufSockBufSendto()** - create a zbuf from a user message and send it to a UDP socket  
**zbufSockRecv()** - receive data in a zbuf from a TCP socket  
**zbufSockRecvfrom()** - receive a message in a zbuf from a UDP socket

**DESCRIPTION**    This library contains routines that communicate over BSD sockets using the *zbuf interface* described in the **zbufLib** manual page. These zbuf socket calls communicate over BSD sockets in a similar manner to the socket routines in **sockLib**, but they avoid copying data unnecessarily between application buffers and network buffers.

**VXWORKS AE PROTECTION DOMAINS**

Under VxWorks AE, this feature is accessible from the kernel protection domain only. This restriction does not apply under non-AE versions of VxWorks.

To use this feature, include the **INCLUDE\_ZBUF SOCK** component.

**SEE ALSO**        **zbufLib, sockLib**



# 2

## *Routines*

<b>a0()</b>	– return the contents of register a0 (also a1 - a7) (68K) .....	403
<b>abort()</b>	– cause abnormal program termination (ANSI) .....	403
<b>abs()</b>	– compute the absolute value of an integer (ANSI).....	404
<b>accept()</b>	– accept a connection from a socket .....	404
<b>acos()</b>	– compute an arc cosine (ANSI) .....	405
<b>acosh()</b>	– compute an arc cosine (ANSI) .....	406
<b>aioPxLibInit()</b>	– initialize the asynchronous I/O (AIO) library .....	406
<b>aioShow()</b>	– show AIO requests .....	407
<b>aioSysInit()</b>	– initialize the AIO system driver .....	407
<b>aio_error()</b>	– retrieve error status of asynchronous I/O operation (POSIX).....	408
<b>aio_read()</b>	– initiate an asynchronous read (POSIX) .....	408
<b>aio_return()</b>	– retrieve return status of asynchronous I/O operation (POSIX) .....	409
<b>aio_suspend()</b>	– wait for asynchronous I/O request(s) (POSIX).....	410
<b>aio_write()</b>	– initiate an asynchronous write (POSIX) .....	410
<b>alarm()</b>	– set an alarm clock for delivery of a signal.....	411
<b>arpAdd()</b>	– create or modify an ARP table entry .....	412
<b>arpDelete()</b>	– remove an ARP table entry .....	413
<b>arpFlush()</b>	– flush all entries in the system ARP table.....	413
<b>arpResolve()</b>	– resolve a hardware address for a specified Internet address.....	414
<b>arpShow()</b>	– display entries in the system ARP table.....	415
<b>arptabShow()</b>	– display the known ARP entries.....	415
<b>asctime()</b>	– convert broken-down time into a string (ANSI).....	416
<b>asctime_r()</b>	– convert broken-down time into a string (POSIX) .....	416
<b>asin()</b>	– compute an arc sine (ANSI) .....	417
<b>asinh()</b>	– compute an arc sine (ANSI) .....	417
<b>assert()</b>	– put diagnostics into programs (ANSI) .....	418
<b>atan()</b>	– compute an arc tangent (ANSI).....	418
<b>atan2()</b>	– compute the arc tangent of y/x (ANSI) .....	419
<b>atan2f()</b>	– compute the arc tangent of y/x (ANSI) .....	420
<b>atanf()</b>	– compute an arc tangent (ANSI).....	420

<b>atexit()</b>	– call a function at program termination (Unimplemented) (ANSI) .....	421
<b>atof()</b>	– convert a string to a double (ANSI) .....	421
<b>atoi()</b>	– convert a string to an int (ANSI) .....	422
<b>atol()</b>	– convert a string to a long (ANSI).....	422
<b>attrib()</b>	– modify MS-DOS file attributes on a file or directory .....	423
<b>b()</b>	– set or display breakpoints.....	424
<b>bcmp()</b>	– compare one buffer to another.....	425
<b>bcopy()</b>	– copy one buffer to another .....	425
<b>bcopyBytes()</b>	– copy one buffer to another one byte at a time .....	426
<b>bcopyLongs()</b>	– copy one buffer to another one long word at a time .....	426
<b>bcopyWords()</b>	– copy one buffer to another one word at a time .....	427
<b>bd()</b>	– delete a breakpoint .....	427
<b>bdall()</b>	– delete all breakpoints .....	428
<b>bfill()</b>	– fill a buffer with a specified character .....	428
<b>bfillBytes()</b>	– fill buffer with a specified character one byte at a time .....	429
<b>bh()</b>	– set a hardware breakpoint.....	429
<b>bind()</b>	– bind a name to a socket.....	430
<b>bindresvport()</b>	– bind a socket to a privileged IP port.....	431
<b>binvert()</b>	– invert the order of bytes in a buffer .....	431
<b>bootBpAnchorExtract()</b>	– extract a backplane address from a device field .....	432
<b>bootChange()</b>	– change the boot line.....	432
<b>bootLeaseExtract()</b>	– extract the lease information from an Internet address .....	433
<b>bootNetmaskExtract()</b>	– extract the net mask field from an Internet address .....	434
<b>bootParamsPrompt()</b>	– prompt for boot line parameters .....	435
<b>bootParamsShow()</b>	– display boot line parameters.....	435
<b>bootpLibInit()</b>	– BOOTP client library initialization.....	436
<b>bootpMsgGet()</b>	– send a BOOTP request message and retrieve reply.....	436
<b>bootpParamsGet()</b>	– retrieve boot parameters using BOOTP .....	437
<b>bootStringToStruct()</b>	– interpret the boot parameters from the boot line.....	441
<b>bootStructToString()</b>	– construct a boot line .....	441
<b>bpfDevCreate()</b>	– create Berkeley Packet Filter device .....	442
<b>bpfDevDelete()</b>	– destroy Berkeley Packet Filter device .....	442
<b>bpfDrv()</b>	– initialize the BPF driver .....	443
<b>bsearch()</b>	– perform a binary search (ANSI) .....	443
<b>bswap()</b>	– swap buffers .....	444
<b>bzero()</b>	– zero out a buffer .....	444
<b>c()</b>	– continue from a breakpoint.....	445
<b>cache4kcLibInit()</b>	– initialize the 4kc cache library.....	445
<b>cacheArchClearEntry()</b>	– clear an entry from a cache (68K, x86) .....	446
<b>cacheArchLibInit()</b>	– initialize the cache library.....	447
<b>cacheAuLibInit()</b>	– initialize the Au cache library .....	449
<b>cacheClear()</b>	– clear all or some entries from a cache .....	449
<b>cacheCy604ClearLine()</b>	– clear a line from a CY7C604 cache.....	450
<b>cacheCy604ClearPage()</b>	– clear a page from a CY7C604 cache.....	450
<b>cacheCy604ClearRegion()</b>	– clear a region from a CY7C604 cache.....	451

<code>cacheCy604ClearSegment()</code>	– clear a segment from a CY7C604 cache .....	451
<code>cacheCy604LibInit()</code>	– initialize the Cypress CY7C604 cache library .....	452
<code>cacheDisable()</code>	– disable the specified cache.....	452
<code>cacheDmaFree()</code>	– free the buffer acquired with <code>cacheDmaMalloc()</code> .....	453
<code>cacheDmaMalloc()</code>	– allocate a cache-safe buffer for DMA devices and drivers .....	453
<code>cacheDrvFlush()</code>	– flush the data cache for drivers.....	454
<code>cacheDrvInvalidate()</code>	– invalidate data cache for drivers .....	454
<code>cacheDrvPhysToVirt()</code>	– translate a physical address for drivers.....	455
<code>cacheDrvVirtToPhys()</code>	– translate a virtual address for drivers.....	455
<code>cacheEnable()</code>	– enable the specified cache.....	456
<code>cacheFlush()</code>	– flush all or some of a specified cache .....	456
<code>cacheInvalidate()</code>	– invalidate all or some of a specified cache .....	457
<code>cacheLibInit()</code>	– initialize the cache library for a processor architecture .....	457
<code>cacheLock()</code>	– lock all or part of a specified cache .....	458
<code>cacheMb930ClearLine()</code>	– clear a line from an MB86930 cache .....	458
<code>cacheMb930LibInit()</code>	– initialize the Fujitsu MB86930 cache library .....	459
<code>cacheMb930LockAuto()</code>	– enable MB86930 automatic locking of kernel instructions/data .....	459
<code>cachePipeFlush()</code>	– flush processor write buffers to memory .....	460
<code>cacheR3kLibInit()</code>	– initialize the R3000 cache library .....	460
<code>cacheR4kLibInit()</code>	– initialize the R4000 cache library .....	461
<code>cacheR5kLibInit()</code>	– initialize the R5000 cache library .....	461
<code>cacheR7kLibInit()</code>	– initialize the R7000 cache library .....	462
<code>cacheR10kLibInit()</code>	– initialize the R10000 cache library .....	463
<code>cacheR32kLibInit()</code>	– initialize the RC32364 cache library .....	463
<code>cacheR32kMalloc()</code>	– allocate a cache-safe buffer, if possible .....	464
<code>cacheR33kLibInit()</code>	– initialize the R33000 cache library .....	464
<code>cacheR333x0LibInit()</code>	– initialize the R333x0 cache library .....	465
<code>cacheSh7040LibInit()</code>	– initialize the SH7040 cache library .....	465
<code>cacheSh7604LibInit()</code>	– initialize the SH7604/SH7615 cache library .....	466
<code>cacheSh7622LibInit()</code>	– initialize the SH7622 cache library .....	466
<code>cacheSh7700LibInit()</code>	– initialize the SH7700 cache library .....	467
<code>cacheSh7729LibInit()</code>	– initialize the SH7729 cache library .....	468
<code>cacheSh7750LibInit()</code>	– initialize the SH7750 cache library .....	469
<code>cacheStoreBufDisable()</code>	– disable the store buffer (MC68060 only).....	470
<code>cacheStoreBufEnable()</code>	– enable the store buffer (MC68060 only).....	470
<code>cacheSun4ClearContext()</code>	– clear a specific context from a Sun-4 cache .....	470
<code>cacheSun4ClearLine()</code>	– clear a line from a Sun-4 cache.....	471
<code>cacheSun4ClearPage()</code>	– clear a page from a Sun-4 cache.....	471
<code>cacheSun4ClearSegment()</code>	– clear a segment from a Sun-4 cache .....	472
<code>cacheSun4LibInit()</code>	– initialize the Sun-4 cache library .....	472
<code>cacheTextUpdate()</code>	– synchronize the instruction and data caches .....	473
<code>cacheTiTms390LibInit()</code>	– initialize the TI TMS390 cache library.....	473
<code>cacheTiTms390PhysToVirt()</code>	– translate a physical address for drivers.....	474
<code>cacheTiTms390VirtToPhys()</code>	– translate a virtual address for <code>cacheLib</code> .....	474
<code>cacheTx49LibInit()</code>	– initialize the Tx49 cache library .....	475

<code>cacheUnlock()</code>	– unlock all or part of a specified cache.....	475
<code>calloc()</code>	– allocate space for an array (ANSI).....	476
<code>cbioBlkCopy()</code>	– block to block (sector to sector) transfer routine.....	476
<code>cbioBlkRW()</code>	– transfer blocks to or from memory.....	477
<code>cbioBytesRW()</code>	– transfer bytes to or from memory.....	477
<code>cbioDevCreate()</code>	– initialize a CBIO device (Generic).....	478
<code>cbioDevVerify()</code>	– verify CBIO_DEV_ID.....	478
<code>cbioIoctl()</code>	– perform ioctl operation on device.....	479
<code>cbioLibInit()</code>	– Initialize CBIO Library.....	480
<code>cbioLock()</code>	– obtain CBIO device semaphore.....	480
<code>cbioModeGet()</code>	– return the mode setting for CBIO device.....	481
<code>cbioModeSet()</code>	– set mode for CBIO device.....	481
<code>cbioParamsGet()</code>	– fill in <b>CBIO_PARAMS</b> structure with CBIO device parameters.....	482
<code>cbioRdyChgdGet()</code>	– determine ready status of CBIO device.....	482
<code>cbioRdyChgdSet()</code>	– force a change in ready status of CBIO device.....	483
<code>cbioShow()</code>	– print information about a CBIO device.....	484
<code>cbioUnlock()</code>	– release CBIO device semaphore.....	484
<code>cbioWrapBlkDev()</code>	– create CBIO wrapper atop a <b>BLK_DEV</b> device.....	485
<code>cbrt()</code>	– compute a cube root.....	485
<code>cbrtf()</code>	– compute a cube root.....	486
<code>cd()</code>	– change the default directory.....	486
<code>cdromFsDevCreate()</code>	– create a <b>cdromFsLib</b> device.....	488
<code>cdromFsInit()</code>	– initialize <b>cdromFsLib</b> .....	488
<code>cdromFsVolConfigShow()</code>	– show the volume configuration information.....	489
<code>ceil()</code>	– compute smallest integer greater than or equal to specified value (ANSI).....	489
<code>ceilf()</code>	– compute smallest integer greater than or equal to specified value (ANSI).....	490
<code>cfree()</code>	– free a block of memory.....	490
<code>chdir()</code>	– set the current default path.....	491
<code>checkStack()</code>	– print a summary of each task’s stack usage.....	491
<code>chkdsk()</code>	– perform consistency checking on a MS-DOS file system.....	492
<code>cleanUpStoreBuffer()</code>	– clean up store buffer after a data store error interrupt.....	493
<code>clearerr()</code>	– clear end-of-file and error flags for a stream (ANSI).....	493
<code>clock()</code>	– determine the processor time in use (ANSI).....	494
<code>clock_getres()</code>	– get the clock resolution (POSIX).....	494
<code>clock_gettime()</code>	– get the current time of the clock (POSIX).....	495
<code>clock_setres()</code>	– set the clock resolution.....	495
<code>clock_settime()</code>	– set the clock to a specified time (POSIX).....	496
<code>close()</code>	– close a file.....	496
<code>closedir()</code>	– close a directory (POSIX).....	497
<code>connect()</code>	– initiate a connection to a socket.....	497
<code>connectWithTimeout()</code>	– attempt socket connection within a specified duration.....	498
<code>copy()</code>	– copy in (or stdin) to out (or stdout).....	498
<code>copyStreams()</code>	– copy from/to specified streams.....	499
<code>cos()</code>	– compute a cosine (ANSI).....	500
<code>cosf()</code>	– compute a cosine (ANSI).....	500

<code>cosh()</code>	– compute a hyperbolic cosine (ANSI) .....	501
<code>coshf()</code>	– compute a hyperbolic cosine (ANSI) .....	501
<code>cp()</code>	– copy file into other file/directory.....	502
<code>cplusCallNewHandler()</code>	– call the allocation failure handler (C++).....	502
<code>cplusCtors()</code>	– call static constructors (C++).....	503
<code>cplusCtorsLink()</code>	– call all linked static constructors (C++) .....	504
<code>cplusDemanglerSet()</code>	– change C++ demangling mode (C++) .....	504
<code>cplusDemanglerStyleSet()</code>	– change C++ demangling style (C++) .....	505
<code>cplusDtors()</code>	– call static destructors (C++).....	505
<code>cplusDtorsLink()</code>	– call all linked static destructors (C++) .....	506
<code>cplusLibInit()</code>	– initialize the C++ library (C++).....	507
<code>cplusXtorSet()</code>	– change C++ static constructor calling strategy (C++) .....	507
<code>cpsr()</code>	– return the contents of the current processor status register (ARM).....	508
<code>creat()</code>	– create a file .....	508
<code>cret()</code>	– continue until the current subroutine returns .....	509
<code>ctime()</code>	– convert time in seconds into a string (ANSI).....	510
<code>ctime_r()</code>	– convert time in seconds into a string (POSIX) .....	510
<code>d()</code>	– display memory .....	512
<code>d0()</code>	– return the contents of register d0 (also d1 - d7) (68K) .....	512
<code>dbgBpTypeBind()</code>	– bind a breakpoint handler to a breakpoint type (MIPS) .....	513
<code>dbgHelp()</code>	– display debugging help menu .....	513
<code>dbgInit()</code>	– initialize the local debugging package.....	514
<code>dcacheDevCreate()</code>	– create a disk cache.....	514
<code>dcacheDevDisable()</code>	– disable the disk cache for this device .....	515
<code>dcacheDevEnable()</code>	– re-enable the disk cache .....	516
<code>dcacheDevMemResize()</code>	– set a new size to a disk cache device .....	516
<code>dcacheDevTune()</code>	– modify tunable disk cache parameters .....	517
<code>dcacheHashTest()</code>	– test hash table integrity .....	518
<code>dcacheShow()</code>	– print information about disk cache .....	519
<code>devs()</code>	– list all system-known devices.....	519
<code>dhcpcBind()</code>	– obtain a set of network configuration parameters with DHCP .....	520
<code>dhcpcBootBind()</code>	– initialize the network with DHCP at boot time.....	521
<code>dhcpcBootInformGet()</code>	– obtain additional configuration parameters with DHCP .....	521
<code>dhcpcBootInit()</code>	– set up the DHCP client parameters and data structures.....	522
<code>dhcpcCacheHookAdd()</code>	– add a routine to store and retrieve lease data.....	523
<code>dhcpcCacheHookDelete()</code>	– delete a lease data storage routine.....	524
<code>dhcpcEventHookAdd()</code>	– add a routine to handle configuration parameters .....	525
<code>dhcpcEventHookDelete()</code>	– remove the configuration parameters handler .....	526
<code>dhcpcInformGet()</code>	– obtain additional configuration parameters with DHCP .....	526
<code>dhcpcInit()</code>	– assign network interface and setup lease request.....	527
<code>dhcpcLibInit()</code>	– DHCP client library initialization.....	528
<code>dhcpcOptionAdd()</code>	– add an option to the client messages .....	529
<code>dhcpcOptionGet()</code>	– retrieve an option provided to a client and store in a buffer.....	530
<code>dhcpcOptionSet()</code>	– add an option to the option request list.....	531
<code>dhcpcParamsGet()</code>	– retrieve current configuration parameters .....	533

<code>dhcpcParamsShow()</code>	– display current lease parameters .....	534
<code>dhcpcRelease()</code>	– relinquish specified lease.....	534
<code>dhcpcServerGet()</code>	– retrieve the current DHCP server .....	535
<code>dhcpcServerShow()</code>	– display current DHCP server .....	535
<code>dhcpcShowInit()</code>	– initialize the DHCP show facility .....	536
<code>dhcpcShutdown()</code>	– disable DHCP client library .....	536
<code>dhcpcTimerGet()</code>	– retrieve current lease timers.....	537
<code>dhcpcTimersShow()</code>	– display current lease timers .....	537
<code>dhcpcVerify()</code>	– renew an established lease .....	538
<code>dhcpsAddressHookAdd()</code>	– assign a permanent address storage hook for the server .....	538
<code>dhcpsInit()</code>	– set up the DHCP server parameters and data structures .....	540
<code>dhcpsLeaseEntryAdd()</code>	– add another entry to the address pool .....	540
<code>dhcpsLeaseHookAdd()</code>	– assign a permanent lease storage hook for the server.....	541
<code>diffTime()</code>	– compute the difference between two calendar times (ANSI) .....	542
<code>dirList()</code>	– list contents of a directory (multi-purpose) .....	542
<code>diskFormat()</code>	– format a disk.....	543
<code>diskInit()</code>	– initialize a file system on a block device .....	544
<code>distCtl()</code>	– perform a distributed objects control function (VxFusion).....	544
<code>distIfShow()</code>	– display information about installed interface adapter (VxFusion).....	548
<code>distInit()</code>	– initialize and bootstrap the current node (VxFusion) .....	549
<code>distNameAdd()</code>	– add an entry to the distributed name database (VxFusion).....	551
<code>distNameFilterShow()</code>	– display the distributed name database filtered by type (VxFusion).....	552
<code>distNameFind()</code>	– find an object by name in the local database (VxFusion) .....	553
<code>distNameFindByValueAndType()</code>	– look up name of object by value and type (VxFusion) .....	554
<code>distNameRemove()</code>	– remove an entry from the distributed name database (VxFusion) .....	555
<code>distNameShow()</code>	– display the entire distributed name database (VxFusion).....	555
<code>distTBufAlloc()</code>	– allocate a telegram buffer from the pool of buffers (VxFusion) .....	556
<code>distTBufFree()</code>	– return a telegram buffer to the pool of buffers (VxFusion).....	557
<code>div()</code>	– compute a quotient and remainder (ANSI).....	557
<code>div_r()</code>	– compute a quotient and remainder (reentrant).....	558
<code>dosFsChkDsk()</code>	– make volume integrity checking.....	558
<code>dosFsDevCreate()</code>	– create file system device .....	559
<code>dosFsLastAccessDateEnable()</code>	– enable last access date updating for this volume.....	560
<code>dosFsLibInit()</code>	– prepare to use the dosFs library .....	560
<code>dosFsShow()</code>	– display dosFs volume configuration data.....	561
<code>dosFsVolDescGet()</code>	– convert a device name into a DOS volume descriptor pointer.....	561
<code>dosFsVolFormat()</code>	– format an MS-DOS compatible volume .....	562
<code>dosSetVolCaseSens()</code>	– set case sensitivity of volume.....	563
<code>dpartDevCreate()</code>	– initialize a partitioned disk .....	564
<code>dpartPartGet()</code>	– retrieve handle for a partition.....	565
<code>dspInit()</code>	– initialize DSP support .....	565
<code>dspShowInit()</code>	– initialize the DSP show facility .....	566
<code>dspTaskRegsShow()</code>	– print the contents of a task’s DSP registers.....	566
<code>e()</code>	– set or display eventpoints (WindView).....	567
<code>edi()</code>	– return the contents of register edi (also esi - eax) (x86/SimNT).....	568

<code>eflags()</code>	– return the contents of the status register (x86/SimNT).....	568
<code>endFindByName()</code>	– find a device using its string name .....	569
<code>envLibInit()</code>	– initialize environment variable facility .....	569
<code>envPrivateCreate()</code>	– create a private environment .....	570
<code>envPrivateDestroy()</code>	– destroy a private environment .....	570
<code>envShow()</code>	– display the environment for a task .....	571
<code>errnoGet()</code>	– get the error status value of the calling task.....	571
<code>errnoOfTaskGet()</code>	– get the error status value of a specified task .....	572
<code>errnoOfTaskSet()</code>	– set the error status value of a specified task.....	572
<code>errnoSet()</code>	– set the error status value of the calling task .....	573
<code>etherMultiAdd()</code>	– add multicast address to a multicast address list .....	573
<code>etherMultiDel()</code>	– delete an Ethernet multicast address record .....	574
<code>etherMultiGet()</code>	– retrieve a table of multicast addresses from a driver .....	574
<code>eventClear()</code>	– clear all events for current task.....	575
<code>eventReceive()</code>	– wait for event(s) .....	575
<code>eventSend()</code>	– send event(s) .....	577
<code>excConnect()</code>	– connect a C routine to an exception vector (PowerPC) .....	577
<code>excCrtConnect()</code>	– connect a C routine to a critical exception vector (PowerPC 403).....	578
<code>excHookAdd()</code>	– specify a routine to be called with exceptions.....	579
<code>excInit()</code>	– initialize the exception handling package .....	580
<code>excIntConnect()</code>	– connect a C routine to an asynchronous exception vector (PowerPC, ARM) .....	580
<code>excIntCrtConnect()</code>	– connect a C routine to a critical interrupt vector (PowerPC 403).....	581
<code>excTask()</code>	– handle task-level exceptions .....	582
<code>excVecGet()</code>	– get a CPU exception vector (PowerPC, ARM) .....	582
<code>excVecInit()</code>	– initialize the exception/interrupt vectors.....	582
<code>excVecSet()</code>	– set a CPU exception vector (PowerPC, ARM).....	584
<code>exit()</code>	– exit a task (ANSI).....	584
<code>exp()</code>	– compute an exponential value (ANSI) .....	585
<code>expf()</code>	– compute an exponential value (ANSI) .....	585
<code>fabs()</code>	– compute an absolute value (ANSI) .....	586
<code>fabsf()</code>	– compute an absolute value (ANSI).....	586
<code>fclose()</code>	– close a stream (ANSI).....	587
<code>fdopen()</code>	– open a file specified by a file descriptor (POSIX) .....	587
<code>fdprintf()</code>	– write a formatted string to a file descriptor.....	588
<code>feof()</code>	– test the end-of-file indicator for a stream (ANSI) .....	588
<code>ferror()</code>	– test the error indicator for a file pointer (ANSI) .....	589
<code>fflush()</code>	– flush a stream (ANSI) .....	589
<code>fgetc()</code>	– return the next character from a stream (ANSI) .....	590
<code>fgetpos()</code>	– store the current value of the file position indicator for a stream (ANSI) .....	590
<code>fgets()</code>	– read a specified number of characters from a stream (ANSI) .....	591
<code>fileno()</code>	– return the file descriptor for a stream (POSIX) .....	591
<code>fileUploadPathClose()</code>	– close the event-destination file (WindView) .....	592
<code>fileUploadPathCreate()</code>	– create a file for depositing event data (Windview) .....	592
<code>fileUploadPathLibInit()</code>	– initialize the <b>wvFileUploadPathLib</b> library (Windview) .....	593
<code>fileUploadPathWrite()</code>	– write to the event-destination file (WindView) .....	593

<b>fioFormatV()</b>	– convert a format string .....	594
<b>fioLibInit()</b>	– initialize the formatted I/O support library .....	595
<b>fioRdString()</b>	– read a string from a file .....	595
<b>fioRead()</b>	– read a buffer .....	596
<b>floatInit()</b>	– initialize floating-point I/O support .....	596
<b>floor()</b>	– compute the largest integer less than or equal to a specified value (ANSI).....	597
<b>floorf()</b>	– compute the largest integer less than or equal to a specified value (ANSI).....	597
<b>fmod()</b>	– compute the remainder of x/y (ANSI) .....	598
<b>fmodf()</b>	– compute the remainder of x/y (ANSI) .....	598
<b>fopen()</b>	– open a file specified by name (ANSI).....	599
<b>fppInit()</b>	– initialize floating-point coprocessor support .....	600
<b>fppProbe()</b>	– probe for the presence of a floating-point coprocessor .....	601
<b>fppRestore()</b>	– restore the floating-point coprocessor context .....	601
<b>fppSave()</b>	– save the floating-point coprocessor context .....	603
<b>fppShowInit()</b>	– initialize the floating-point show facility .....	604
<b>fppTaskRegsGet()</b>	– get the floating-point registers from a task TCB.....	605
<b>fppTaskRegsSet()</b>	– set the floating-point registers of a task .....	605
<b>fppTaskRegsShow()</b>	– print the contents of a task’s floating-point registers .....	606
<b>fprintf()</b>	– write a formatted string to a stream (ANSI) .....	606
<b>fputc()</b>	– write a character to a stream (ANSI) .....	610
<b>fputs()</b>	– write a string to a stream (ANSI) .....	611
<b>fread()</b>	– read data into an array (ANSI).....	611
<b>free()</b>	– free a block of memory (ANSI) .....	612
<b>freopen()</b>	– open a file specified by name (ANSI).....	612
<b>frexp()</b>	– break floating-point number into normalized fraction and power of 2 (ANSI) .....	613
<b>fscanf()</b>	– read and convert characters from a stream (ANSI) .....	614
<b>fseek()</b>	– set the file position indicator for a stream (ANSI).....	618
<b>fsetpos()</b>	– set the file position indicator for a stream (ANSI).....	619
<b>fstat()</b>	– get file status information (POSIX).....	619
<b>fstatfs()</b>	– get file status information (POSIX).....	620
<b>ftell()</b>	– return the current value of the file position indicator for a stream (ANSI).....	620
<b>ftpCommand()</b>	– send an FTP command and get the reply .....	621
<b>ftpCommandEnhanced()</b>	– send an FTP command and get the complete RFC reply code .....	622
<b>ftpDataConnGet()</b>	– get a completed FTP data connection .....	623
<b>ftpDataConnInit()</b>	– initialize an FTP data connection using PORT mode .....	623
<b>ftpDataConnInitPassiveMode()</b>	– initialize an FTP data connection using PASV mode.....	624
<b>ftpdDelete()</b>	– terminate the FTP server task.....	625
<b>ftpdInit()</b>	– initialize the FTP server task.....	625
<b>ftpHookup()</b>	– get a control connection to the FTP server on a specified host.....	626
<b>ftpLibDebugOptionSet()</b>	– set the debug level of the ftp library routines .....	626
<b>ftpLogin()</b>	– log in to a remote FTP server.....	627
<b>ftpLs()</b>	– list directory contents via FTP .....	627
<b>ftpReplyGet()</b>	– get an FTP command reply .....	628
<b>ftpReplyGetEnhanced()</b>	– get an FTP command reply .....	629
<b>ftpTransientConfigGet()</b>	– get parameters for host FTP_TRANSIENT responses.....	630

<b>ftpTransientConfigSet()</b>	– set parameters for host FTP_TRANSIENT responses.....	630
<b>ftpTransientFatalInstall()</b>	– set applette to stop FTP transient host responses.....	631
<b>ftpXfer()</b>	– initiate a transfer via FTP .....	631
<b>ftruncate()</b>	– truncate a file (POSIX) .....	633
<b>fwrite()</b>	– write from a specified array (ANSI) .....	633
<b>getc()</b>	– return the next character from a stream (ANSI) .....	635
<b>getchar()</b>	– return the next character from the standard input stream (ANSI).....	635
<b>getcwd()</b>	– get the current default path (POSIX) .....	636
<b>getenv()</b>	– get an environment variable (ANSI) .....	636
<b>gethostname()</b>	– get the symbolic name of this machine.....	637
<b>getpeername()</b>	– get the name of a connected peer.....	637
<b>gets()</b>	– read characters from the standard input stream (ANSI).....	638
<b>getsockname()</b>	– get a socket name .....	638
<b>getsockopt()</b>	– get socket options .....	639
<b>getw()</b>	– read the next word (32-bit integer) from a stream .....	640
<b>getwd()</b>	– get the current default path .....	640
<b>gmtime()</b>	– convert calendar time into UTC broken-down time (ANSI) .....	641
<b>gmtime_r()</b>	– convert calendar time into broken-down time (POSIX).....	641
<b>h()</b>	– display or set the size of shell history .....	643
<b>hashFuncIterScale()</b>	– iterative scaling hashing function for strings.....	643
<b>hashFuncModulo()</b>	– hashing function using remainder technique .....	644
<b>hashFuncMultiply()</b>	– multiplicative hashing function .....	644
<b>hashKeyCmp()</b>	– compare keys as 32 bit identifiers .....	645
<b>hashKeyStrCmp()</b>	– compare keys based on strings they point to.....	645
<b>hashLibInit()</b>	– initialize hash table library .....	646
<b>hashTblCreate()</b>	– create a hash table .....	646
<b>hashTblDelete()</b>	– delete a hash table .....	647
<b>hashTblDestroy()</b>	– destroy a hash table .....	647
<b>hashTblEach()</b>	– call a routine for each node in a hash table .....	648
<b>hashTblFind()</b>	– find a hash node that matches the specified key .....	648
<b>hashTblInit()</b>	– initialize a hash table .....	649
<b>hashTblPut()</b>	– put a hash node into the specified hash table .....	649
<b>hashTblRemove()</b>	– remove a hash node from a hash table.....	650
<b>hashTblTerminate()</b>	– terminate a hash table.....	650
<b>help()</b>	– print a synopsis of selected routines .....	651
<b>hostAdd()</b>	– add a host to the host table .....	652
<b>hostDelete()</b>	– delete a host from the host table .....	652
<b>hostGetByAddr()</b>	– look up a host in the host table by its Internet address .....	653
<b>hostGetByName()</b>	– look up a host in the host table by its name .....	654
<b>hostShow()</b>	– display the host table.....	654
<b>hostTblInit()</b>	– initialize the network host table.....	655
<b>i()</b>	– print a summary of each task's TCB.....	656
<b>iam()</b>	– set the remote user name and password .....	657
<b>icmpShowInit()</b>	– initialize ICMP show routines.....	657
<b>icmpstatShow()</b>	– display statistics for ICMP .....	658

<code>ifAddrAdd()</code>	– add an interface address for a network interface .....	658
<code>ifAddrDelete()</code>	– delete an interface address for a network interface .....	659
<code>ifAddrGet()</code>	– get the Internet address of a network interface .....	659
<code>ifAddrSet()</code>	– set an interface address for a network interface.....	660
<code>ifAllRoutesDelete()</code>	– delete all routes associated with a network interface .....	660
<code>ifBroadcastGet()</code>	– get the broadcast address for a network interface .....	661
<code>ifBroadcastSet()</code>	– set the broadcast address for a network interface.....	661
<code>ifDstAddrGet()</code>	– get the Internet address of a point-to-point peer .....	662
<code>ifDstAddrSet()</code>	– define an address for the other end of a point-to-point link .....	663
<code>ifFlagChange()</code>	– change the network interface flags.....	663
<code>ifFlagGet()</code>	– get the network interface flags.....	664
<code>ifFlagSet()</code>	– specify the flags for a network interface.....	664
<code>ifIndexAlloc()</code>	– return a unique interface index.....	665
<code>ifIndexLibInit()</code>	– initializes library variables.....	666
<code>ifIndexLibShutdown()</code>	– frees library variables .....	666
<code>ifIndexTest()</code>	– returns true if an index has been allocated. ....	667
<code>ifIndexToIfName()</code>	– returns the interface name given the interface index .....	667
<code>ifMaskGet()</code>	– get the subnet mask for a network interface .....	668
<code>ifMaskSet()</code>	– define a subnet for a network interface .....	668
<code>ifMetricGet()</code>	– get the metric for a network interface .....	669
<code>ifMetricSet()</code>	– specify a network interface hop count.....	669
<code>ifNameToIfIndex()</code>	– returns the interface index given the interface name .....	670
<code>ifRouteDelete()</code>	– delete routes associated with a network interface .....	670
<code>ifShow()</code>	– display the attached network interfaces.....	671
<code>ifunit()</code>	– map an interface name to an interface structure pointer .....	671
<code>ifUnnumberedSet()</code>	– configure an interface to be unnumbered .....	672
<code>igmpShowInit()</code>	– initialize IGMP show routines .....	673
<code>igmpstatShow()</code>	– display statistics for IGMP.....	674
<code>index()</code>	– find the first occurrence of a character in a string.....	674
<code>inet_addr()</code>	– convert a dot notation Internet address to a long integer.....	675
<code>inet_aton()</code>	– convert a network address from dot notation, store in a structure .....	675
<code>inet_lnaof()</code>	– get the local address (host number) from the Internet address .....	676
<code>inet_makeaddr()</code>	– form an Internet address from network and host numbers .....	676
<code>inet_makeaddr_b()</code>	– form an Internet address from network and host numbers .....	677
<code>inet_netof()</code>	– return the network number from an Internet address .....	678
<code>inet_netof_string()</code>	– extract the network address in dot notation .....	678
<code>inet_network()</code>	– convert an Internet network number from string to address.....	679
<code>inet_ntoa()</code>	– convert a network address to dotted decimal notation.....	679
<code>inet_ntoa_b()</code>	– convert an network address to dot notation, store it in a buffer.....	680
<code>inetstatShow()</code>	– display all active connections for Internet protocol sockets .....	681
<code>infinity()</code>	– return a very large double .....	681
<code>infinityf()</code>	– return a very large float.....	682
<code>inflate()</code>	– inflate compressed code.....	682
<code>intConnect()</code>	– connect a C routine to a hardware interrupt.....	683
<code>intContext()</code>	– determine if the current state is in interrupt or task context .....	686

<b>intCount()</b>	– get the current interrupt nesting depth.....	686
<b>intCRGet()</b>	– read the contents of the cause register (MIPS) .....	686
<b>intCRSet()</b>	– write the contents of the cause register (MIPS).....	687
<b>intDisable()</b>	– disable corresponding interrupt bits (MIPS, PowerPC, ARM).....	687
<b>intEnable()</b>	– enable corresponding interrupt bits (MIPS, PowerPC, ARM).....	688
<b>intHandlerCreate()</b>	– construct an interrupt handler for a C routine (68K, x86, MIPS, SimSolaris)...	688
<b>intHandlerCreateI86()</b>	– construct an interrupt handler for a C routine (x86).....	689
<b>intLevelSet()</b>	– set the interrupt level (68K, x86, ARM, SimSolaris, SimNT and SH).....	690
<b>intLock()</b>	– lock out interrupts .....	691
<b>intLockLevelGet()</b>	– get current interrupt lock-out level (68K, x86, ARM, SH, SimSolaris, SimNT)	693
<b>intLockLevelSet()</b>	– set current interrupt lock-out level (68K, x86, ARM, SH, SimSolaris, SimNT)	693
<b>intSRGet()</b>	– read the contents of the status register (MIPS).....	694
<b>intSRSet()</b>	– update the contents of the status register (MIPS).....	694
<b>intStackEnable()</b>	– enable or disable the interrupt stack usage (x86).....	695
<b>intUninitVecSet()</b>	– set the uninitialized vector handler (ARM) .....	695
<b>intUnlock()</b>	– cancel interrupt locks .....	696
<b>intVecBaseGet()</b>	– get vector (trap) base address (68K, x86, MIPS, ARM, SimSolaris, SimNT).....	696
<b>intVecBaseSet()</b>	– set vector (trap) base address (68K, x86, MIPS, ARM, SimSolaris, SimNT).....	697
<b>intVecGet()</b>	– get an interrupt vector (68K, x86, MIPS, SH, SimSolaris, SimNT) .....	698
<b>intVecGet2()</b>	– get a CPU vector, gate type(int/trap), and gate selector (x86) .....	699
<b>intVecSet()</b>	– set a CPU vector (trap) (68K, x86, MIPS, SH, SimSolaris, SimNT).....	699
<b>intVecSet2()</b>	– set a CPU vector, gate type(int/trap), and selector (x86) .....	703
<b>intVecTableWriteProtect()</b>	– write-protect exception vector table (68K, x86, ARM, SimSolaris, SimNT)	704
<b>ioctl()</b>	– perform an I/O control function .....	704
<b>ioDefPathGet()</b>	– get the current default path.....	705
<b>ioDefPathSet()</b>	– set the current default path .....	706
<b>ioGlobalStdGet()</b>	– get the file descriptor for global standard input/output/error .....	706
<b>ioGlobalStdSet()</b>	– set the file descriptor for global standard input/output/error.....	707
<b>ioHelp()</b>	– print a synopsis of I/O utility functions .....	707
<b>iosDevAdd()</b>	– add a device to the I/O system .....	708
<b>iosDevDelete()</b>	– delete a device from the I/O system.....	708
<b>iosDevFind()</b>	– find an I/O device in the device list .....	709
<b>iosDevShow()</b>	– display the list of devices in the system .....	709
<b>iosDrvInstall()</b>	– install an I/O driver .....	710
<b>iosDrvRemove()</b>	– remove an I/O driver.....	710
<b>iosDrvShow()</b>	– display a list of system drivers .....	711
<b>iosFdShow()</b>	– display a list of file descriptor names in the system.....	711
<b>iosFdValue()</b>	– validate an open file descriptor and return the driver-specific value .....	711
<b>iosInit()</b>	– initialize the I/O system.....	712
<b>iosShowInit()</b>	– initialize the I/O system show facility .....	712
<b>ioTaskStdGet()</b>	– get the file descriptor for task standard input/output/error .....	713
<b>ioTaskStdSet()</b>	– set the file descriptor for task standard input/output/error.....	713
<b>ipAttach()</b>	– a generic attach routine for the TCP/IP network stack .....	714
<b>ipDetach()</b>	– a generic detach routine for the TCP/IP network stack .....	714
<b>ipFilterHookAdd()</b>	– add a routine to receive all internet protocol packets.....	715

<b>ipFilterHookDelete()</b>	– delete a IP filter hook routine .....	716
<b>ipFilterLibInit()</b>	– initialize IP filter facility .....	716
<b>ipstatShow()</b>	– display IP statistics .....	717
<b>irint()</b>	– convert a double-precision value to an integer .....	717
<b>irintf()</b>	– convert a single-precision value to an integer .....	718
<b>iround()</b>	– round a number to the nearest integer .....	718
<b>iroundf()</b>	– round a number to the nearest integer .....	719
<b>isalnum()</b>	– test whether a character is alphanumeric (ANSI) .....	719
<b>isalpha()</b>	– test whether a character is a letter (ANSI) .....	720
<b>isatty()</b>	– return whether the underlying driver is a tty device .....	720
<b>iscntrl()</b>	– test whether a character is a control character (ANSI) .....	721
<b>isdigit()</b>	– test whether a character is a decimal digit (ANSI) .....	721
<b>isgraph()</b>	– test whether a character is a printing, non-white-space character (ANSI) .....	722
<b>islower()</b>	– test whether a character is a lower-case letter (ANSI) .....	722
<b>isprint()</b>	– test whether a character is printable, including the space character (ANSI) .....	723
<b>ispunct()</b>	– test whether a character is punctuation (ANSI) .....	723
<b>isspace()</b>	– test whether a character is a white-space character (ANSI) .....	724
<b>isupper()</b>	– test whether a character is an upper-case letter (ANSI) .....	724
<b>isxdigit()</b>	– test whether a character is a hexadecimal digit (ANSI) .....	725
<b>kernelInit()</b>	– initialize the kernel .....	726
<b>kernelTimeSlice()</b>	– enable round-robin selection .....	727
<b>kernelVersion()</b>	– return the kernel revision string .....	727
<b>kill()</b>	– send a signal to a task (POSIX) .....	728
<b>l()</b>	– disassemble and display a specified number of instructions .....	729
<b>labs()</b>	– compute the absolute value of a long (ANSI) .....	729
<b>ld()</b>	– load an object module into memory .....	730
<b>ldexp()</b>	– multiply a number by an integral power of 2 (ANSI) .....	731
<b>ldiv()</b>	– compute the quotient and remainder of the division (ANSI) .....	732
<b>ldiv_r()</b>	– compute a quotient and remainder (reentrant) .....	732
<b>ledClose()</b>	– discard the line-editor ID .....	733
<b>ledControl()</b>	– change the line-editor ID parameters .....	733
<b>ledOpen()</b>	– create a new line-editor ID .....	734
<b>ledRead()</b>	– read a line with line-editing .....	734
<b>lio_listio()</b>	– initiate a list of asynchronous I/O requests (POSIX) .....	735
<b>listen()</b>	– enable connections to a socket .....	736
<b>lkAddr()</b>	– list symbols whose values are near a specified value .....	736
<b>lkup()</b>	– list symbols .....	737
<b>ll()</b>	– generate a long listing of directory contents .....	737
<b>llr()</b>	– do a long listing of directory and all its subdirectories contents .....	738
<b>loadModule()</b>	– load an object module into memory .....	739
<b>loadModuleAt()</b>	– load an object module into memory .....	739
<b>localeconv()</b>	– set the components of an object with type lconv (ANSI) .....	742
<b>localtime()</b>	– convert calendar time into broken-down time (ANSI) .....	745
<b>localtime_r()</b>	– convert calendar time into broken-down time (POSIX) .....	746
<b>log()</b>	– compute a natural logarithm (ANSI) .....	746

<b>log2()</b>	– compute a base-2 logarithm.....	747
<b>log2f()</b>	– compute a base-2 logarithm.....	747
<b>log10()</b>	– compute a base-10 logarithm (ANSI).....	748
<b>log10f()</b>	– compute a base-10 logarithm (ANSI).....	748
<b>logf()</b>	– compute a natural logarithm (ANSI).....	749
<b>logFdAdd()</b>	– add a logging file descriptor.....	749
<b>logFdDelete()</b>	– delete a logging file descriptor.....	750
<b>logFdSet()</b>	– set the primary logging file descriptor.....	750
<b>loginDefaultEncrypt()</b>	– default password encryption routine.....	751
<b>loginEncryptInstall()</b>	– install an encryption routine.....	751
<b>loginInit()</b>	– initialize the login table.....	752
<b>logInit()</b>	– initialize message logging library.....	753
<b>loginPrompt()</b>	– display a login prompt and validate a user entry.....	753
<b>loginStringSet()</b>	– change the login string.....	754
<b>loginUserAdd()</b>	– add a user to the login table.....	754
<b>loginUserDelete()</b>	– delete a user entry from the login table.....	755
<b>loginUserShow()</b>	– display the user login table.....	756
<b>loginUserVerify()</b>	– verify a user name and password in the login table.....	756
<b>logMsg()</b>	– log a formatted error message.....	757
<b>logout()</b>	– log out of the VxWorks system.....	758
<b>logTask()</b>	– message-logging support task.....	758
<b>longjmp()</b>	– perform non-local goto by restoring saved environment (ANSI).....	759
<b>ls()</b>	– generate a brief listing of a directory.....	759
<b>lseek()</b>	– set a file read/write pointer.....	760
<b>lsr()</b>	– list the contents of a directory and any of its subdirectories.....	761
<b>lstAdd()</b>	– add a node to the end of a list.....	761
<b>lstConcat()</b>	– concatenate two lists.....	762
<b>lstCount()</b>	– report the number of nodes in a list.....	762
<b>lstDelete()</b>	– delete a specified node from a list.....	763
<b>lstExtract()</b>	– extract a sublist from a list.....	763
<b>lstFind()</b>	– find a node in a list.....	764
<b>lstFirst()</b>	– find first node in list.....	764
<b>lstFree()</b>	– free up a list.....	765
<b>lstGet()</b>	– delete and return the first node from a list.....	765
<b>lstInit()</b>	– initialize a list descriptor.....	766
<b>lstInsert()</b>	– insert a node in a list after a specified node.....	766
<b>lstLast()</b>	– find the last node in a list.....	767
<b>lstLibInit()</b>	– initializes <b>lstLib</b> module.....	767
<b>lstNext()</b>	– find the next node in a list.....	768
<b>lstNStep()</b>	– find a list node nStep steps away from a specified node.....	768
<b>lstNth()</b>	– find the Nth node in a list.....	769
<b>lstPrevious()</b>	– find the previous node in a list.....	769
<b>m()</b>	– modify memory.....	770
<b>m2Delete()</b>	– delete all the MIB-II library groups.....	770
<b>m2IcmpDelete()</b>	– delete all resources used to access the ICMP group.....	771

<b>m2IcmpGroupInfoGet()</b>	– get the MIB-II ICMP-group global variables.....	771
<b>m2IcmpInit()</b>	– initialize MIB-II ICMP-group access.....	772
<b>m2If8023PacketCount()</b>	– increment the packet counters for an 802.3 device.....	772
<b>m2IfAlloc()</b>	– allocate the structure for the interface table.....	773
<b>m2IfCommonValsGet()</b>	– get the common values.....	774
<b>m2IfCounterUpdate()</b>	– increment interface counters.....	775
<b>m2IfCtrUpdateRtnInstall()</b>	– install an interface counter update routine.....	775
<b>m2IfDefaultValsGet()</b>	– get the default values for the counters.....	776
<b>m2IfDelete()</b>	– delete all resources used to access the interface group.....	776
<b>m2IfFree()</b>	– free an interface data structure.....	777
<b>m2IfGenericPacketCount()</b>	– increment the interface packet counters.....	777
<b>m2IfGroupInfoGet()</b>	– get the MIB-II interface-group scalar variables.....	778
<b>m2IfInit()</b>	– initialize MIB-II interface-group routines.....	778
<b>m2IfPktCountRtnInstall()</b>	– install an interface packet counter routine.....	779
<b>m2IfRcvAddrEntryGet()</b>	– get the rcvAddress table entries for a given address.....	779
<b>m2IfRcvAddrEntrySet()</b>	– modify the entries of the rcvAddressTable.....	780
<b>m2IfStackEntryGet()</b>	– get a MIB-II interface-group table entry.....	781
<b>m2IfStackEntrySet()</b>	– modify the status of a relationship.....	781
<b>m2IfStackTblUpdate()</b>	– update the relationship between the sub-layers.....	782
<b>m2IfTableUpdate()</b>	– insert or remove an entry in the ifTable.....	783
<b>m2IfTblEntryGet()</b>	– get a MIB-II interface-group table entry.....	783
<b>m2IfTblEntrySet()</b>	– set the state of a MIB-II interface entry to UP or DOWN.....	784
<b>m2IfVariableUpdate()</b>	– update the contents of an interface non-counter object.....	785
<b>m2IfVarUpdateRtnInstall()</b>	– install an interface variable update routine.....	786
<b>m2Init()</b>	– initialize the SNMP MIB-2 library.....	786
<b>m2IpAddrTblEntryGet()</b>	– get an IP MIB-II address entry.....	787
<b>m2IpAtransTblEntryGet()</b>	– get a MIB-II ARP table entry.....	787
<b>m2IpAtransTblEntrySet()</b>	– add, modify, or delete a MIB-II ARP entry.....	788
<b>m2IpDelete()</b>	– delete all resources used to access the IP group.....	789
<b>m2IpGroupInfoGet()</b>	– get the MIB-II IP-group scalar variables.....	789
<b>m2IpGroupInfoSet()</b>	– set MIB-II IP-group variables to new values.....	790
<b>m2IpInit()</b>	– initialize MIB-II IP-group access.....	790
<b>m2IpRouteTblEntryGet()</b>	– get a MIB-2 routing table entry.....	791
<b>m2IpRouteTblEntrySet()</b>	– set a MIB-II routing table entry.....	791
<b>m2RipDelete()</b>	– delete the RIP MIB support.....	792
<b>m2RipGlobalCountersGet()</b>	– get MIB-II RIP-group global counters.....	793
<b>m2RipIfConfEntryGet()</b>	– get MIB-II RIP-group interface entry.....	793
<b>m2RipIfConfEntrySet()</b>	– set MIB-II RIP-group interface entry.....	794
<b>m2RipIfStatEntryGet()</b>	– get MIB-II RIP-group interface entry.....	794
<b>m2RipInit()</b>	– initialize the RIP MIB support.....	795
<b>m2SysDelete()</b>	– delete resources used to access the MIB-II system group.....	795
<b>m2SysGroupInfoGet()</b>	– get system-group MIB-II variables.....	796
<b>m2SysGroupInfoSet()</b>	– set system-group MIB-II variables to new values.....	796
<b>m2SysInit()</b>	– initialize MIB-II system-group routines.....	797
<b>m2TcpConnEntryGet()</b>	– get a MIB-II TCP connection table entry.....	797

<b>m2TcpConnEntrySet()</b>	– set a TCP connection to the closed state.....	798
<b>m2TcpDelete()</b>	– delete all resources used to access the TCP group .....	798
<b>m2TcpGroupInfoGet()</b>	– get MIB-II TCP-group scalar variables.....	799
<b>m2TcpInit()</b>	– initialize MIB-II TCP-group access .....	799
<b>m2UdpDelete()</b>	– delete all resources used to access the UDP group .....	800
<b>m2UdpGroupInfoGet()</b>	– get MIB-II UDP-group scalar variables.....	800
<b>m2UdpInit()</b>	– initialize MIB-II UDP-group access .....	801
<b>m2UdpTblEntryGet()</b>	– get a UDP MIB-II entry from the UDP list of listeners .....	801
<b>mach()</b>	– return the contents of system register mach (also macl, pr) (SH) .....	802
<b>malloc()</b>	– allocate a block of memory from the system memory partition (ANSI).....	802
<b>mathHardInit()</b>	– initialize hardware floating-point math support.....	803
<b>mathSoftInit()</b>	– initialize software floating-point math support.....	803
<b>mblen()</b>	– calculate the length of a multibyte character (Unimplemented) (ANSI) .....	804
<b>mbstowcs()</b>	– convert series of multibyte char’s to wide char’s (Unimplemented) (ANSI)	804
<b>mbtowc()</b>	– convert multibyte character to a wide character (Unimplemented) (ANSI) .	805
<b>mbufShow()</b>	– report mbuf statistics .....	805
<b>memAddToPool()</b>	– add memory to the system memory partition .....	806
<b>memalign()</b>	– allocate aligned memory .....	806
<b>memchr()</b>	– search a block of memory for a character (ANSI).....	807
<b>memcmp()</b>	– compare two blocks of memory (ANSI) .....	807
<b>memcpy()</b>	– copy memory from one location to another (ANSI).....	808
<b>memDevCreate()</b>	– create a memory device .....	808
<b>memDevCreateDir()</b>	– create a memory device for multiple files.....	810
<b>memDevDelete()</b>	– delete a memory device .....	810
<b>memDrv()</b>	– install a memory driver .....	811
<b>memFindMax()</b>	– find the largest free block in the system memory partition .....	811
<b>memmove()</b>	– copy memory from one location to another (ANSI).....	812
<b>memOptionsSet()</b>	– set the debug options for the system memory partition.....	812
<b>memPartAddToPool()</b>	– add memory to a memory partition .....	813
<b>memPartAlignedAlloc()</b>	– allocate aligned memory from a partition .....	813
<b>memPartAlloc()</b>	– allocate a block of memory from a partition .....	814
<b>memPartCreate()</b>	– create a memory partition .....	814
<b>memPartFindMax()</b>	– find the size of the largest available free block .....	815
<b>memPartFree()</b>	– free a block of memory in a partition .....	815
<b>memPartInfoGet()</b>	– get partition information.....	816
<b>memPartOptionsSet()</b>	– set the debug options for a memory partition .....	816
<b>memPartRealloc()</b>	– reallocate a block of memory in a specified partition .....	817
<b>memPartShow()</b>	– show partition blocks and statistics.....	818
<b>memPartSmCreate()</b>	– create a shared memory partition (VxMP) .....	818
<b>memset()</b>	– set a block of memory (ANSI) .....	819
<b>memShow()</b>	– show system memory partition blocks and statistics .....	820
<b>memShowInit()</b>	– initialize the memory partition show facility .....	821
<b>mkdir()</b>	– make a directory .....	821
<b>mktime()</b>	– convert broken-down time into calendar time (ANSI).....	822
<b>mlock()</b>	– lock specified pages into memory (POSIX) .....	822

<b>mlockall()</b>	– lock all pages used by a process into memory (POSIX).....	823
<b>mmuPhysToVirt()</b>	– translate a physical address to a virtual address (ARM).....	823
<b>mmuPro32LibInit()</b>	– initialize module.....	825
<b>mmuSh7700LibInit()</b>	– initialize module.....	825
<b>mmuSh7750LibInit()</b>	– initialize module.....	826
<b>mmuVirtToPhys()</b>	– translate a virtual address to a physical address (ARM).....	826
<b>modf()</b>	– separate floating-point number into integer and fraction parts (ANSI)	827
<b>moduleCheck()</b>	– verify checksums on all modules.....	827
<b>moduleCreate()</b>	– create and initialize a module.....	828
<b>moduleCreateHookAdd()</b>	– add a routine to be called when a module is added .....	829
<b>moduleCreateHookDelete()</b>	– delete a previously added module create hook routine .....	829
<b>moduleDelete()</b>	– delete module ID information (use <b>unld()</b> to reclaim space).....	830
<b>moduleFindByGroup()</b>	– find a module by group number.....	830
<b>moduleFindByName()</b>	– find a module by name.....	831
<b>moduleFindByNameAndPath()</b>	– find a module by file name and path.....	832
<b>moduleFlagsGet()</b>	– get the flags associated with a module ID.....	832
<b>moduleIdListGet()</b>	– get a list of loaded modules.....	833
<b>moduleInfoGet()</b>	– get information about an object module.....	833
<b>moduleNameGet()</b>	– get the name associated with a module ID.....	834
<b>moduleSegFirst()</b>	– find the first segment in a module.....	834
<b>moduleSegGet()</b>	– get (delete and return) the first segment from a module .....	835
<b>moduleSegNext()</b>	– find the next segment in a module .....	835
<b>moduleShow()</b>	– show the current status for all the loaded modules.....	836
<b>mountdInit()</b>	– initialize the mount daemon.....	836
<b>mqPxLibInit()</b>	– initialize the POSIX message queue library.....	837
<b>mqPxShowInit()</b>	– initialize the POSIX message queue show facility.....	838
<b>mq_close()</b>	– close a message queue (POSIX).....	838
<b>mq_getattr()</b>	– get message queue attributes (POSIX) .....	839
<b>mq_notify()</b>	– notify a task that a message is available on a queue (POSIX).....	840
<b>mq_open()</b>	– open a message queue (POSIX).....	841
<b>mq_receive()</b>	– receive a message from a message queue (POSIX).....	842
<b>mq_send()</b>	– send a message to a message queue (POSIX).....	843
<b>mq_setattr()</b>	– set message queue attributes (POSIX).....	844
<b>mq_unlink()</b>	– remove a message queue (POSIX) .....	845
<b>mRegs()</b>	– modify registers .....	845
<b>mRouteAdd()</b>	– add multiple routes to the same destination.....	846
<b>mRouteDelete()</b>	– delete a route from the routing table.....	847
<b>mRouteEntryAdd()</b>	– add a protocol-specific route to the routing table .....	848
<b>mRouteEntryDelete()</b>	– delete route from the routing table.....	849
<b>mRouteShow()</b>	– display all IP routes (verbose information) .....	849
<b>msgQCreate()</b>	– create and initialize a message queue .....	850
<b>msgQDelete()</b>	– delete a message queue .....	851
<b>msgQDistCreate()</b>	– create a distributed message queue (VxFusion) .....	851
<b>msgQDistGrpAdd()</b>	– add a distributed message queue to a group (VxFusion).....	853
<b>msgQDistGrpDelete()</b>	– delete a distributed message queue from a group (VxFusion).....	854

<b>msgQDistGrpShow()</b>	- display all or one group with its members (VxFusion) .....	854
<b>msgQDistNumMsgs()</b>	- get number of messages in a distributed message queue (VxFusion) .....	855
<b>msgQDistReceive()</b>	- receive a message from a distributed message queue (VxFusion) .....	856
<b>msgQDistSend()</b>	- send a message to a distributed message queue (VxFusion) .....	857
<b>msgQDistShowInit()</b>	- initialize the distributed message queue show package (VxFusion) .....	859
<b>msgQEvStart()</b>	- start event notification process for a message queue .....	859
<b>msgQEvStop()</b>	- stop event notification process for a message queue .....	860
<b>msgQInfoGet()</b>	- get information about a message queue .....	861
<b>msgQNumMsgs()</b>	- get the number of messages queued to a message queue .....	863
<b>msgQReceive()</b>	- receive a message from a message queue .....	864
<b>msgQSend()</b>	- send a message to a message queue .....	865
<b>msgQShow()</b>	- show information about a message queue .....	866
<b>msgQShowInit()</b>	- initialize the message queue show facility .....	867
<b>msgQSmCreate()</b>	- create and initialize a shared memory message queue (VxMP) .....	868
<b>munlock()</b>	- unlock specified pages (POSIX) .....	869
<b>munlockall()</b>	- unlock all pages used by a process (POSIX) .....	869
<b>muxAddressForm()</b>	- form a frame with a link-layer address .....	870
<b>muxAddrResFuncAdd()</b>	- replace the default address resolution function .....	871
<b>muxAddrResFuncDel()</b>	- delete an address resolution function .....	872
<b>muxAddrResFuncGet()</b>	- get the address resolution function for ifType/protocol .....	873
<b>muxBind()</b>	- create a binding between a network service and an END .....	874
<b>muxDevExists()</b>	- tests whether a device is already loaded into the MUX .....	876
<b>muxDevLoad()</b>	- load a driver into the MUX .....	876
<b>muxDevStart()</b>	- start a device by calling its start routine .....	877
<b>muxDevStop()</b>	- stop a device by calling its stop routine .....	878
<b>muxDevUnload()</b>	- unloads a device from the MUX .....	879
<b>muxIoctl()</b>	- send control information to the MUX or to a device .....	880
<b>muxLibInit()</b>	- initialize global state for the MUX .....	881
<b>muxLinkHeaderCreate()</b>	- attach a link-level header to a packet .....	881
<b>muxMCastAddrAdd()</b>	- add a multicast address to a device's multicast table .....	882
<b>muxMCastAddrDel()</b>	- delete a multicast address from a device's multicast table .....	883
<b>muxMCastAddrGet()</b>	- get the multicast address table from the MUX/Driver .....	884
<b>muxPacketAddrGet()</b>	- get addressing information from a packet .....	885
<b>muxPacketDataGet()</b>	- return the data from a packet .....	886
<b>muxPollDevAdd()</b>	- adds a device to list polled by tMuxPollTask .....	887
<b>muxPollDevDel()</b>	- removes a device from the list polled by tMuxPollTask .....	887
<b>muxPollDevStat()</b>	- reports whether device is on list polled by tMuxPollTask .....	888
<b>muxPollEnd()</b>	- shuts down tMuxPollTask and returns devices to interrupt mode .....	888
<b>muxPollReceive()</b>	- now deprecated, see <b>muxTkPollReceive()</b> .....	889
<b>muxPollSend()</b>	- now deprecated, see <b>muxTkPollSend()</b> .....	890
<b>muxPollStart()</b>	- initialize and start the MUX poll task .....	891
<b>muxSend()</b>	- send a packet out on a network interface .....	892
<b>muxShow()</b>	- display configuration of devices registered with the MUX .....	893
<b>muxTaskDelayGet()</b>	- get the delay on the polling task .....	893
<b>muxTaskDelaySet()</b>	- set the inter-cycle delay on the polling task .....	894

<code>muxTaskPriorityGet()</code>	– get the priority of <code>tMuxPollTask</code> .....	894
<code>muxTaskPrioritySet()</code>	– reset the priority of <code>tMuxPollTask</code> .....	895
<code>muxTkBind()</code>	– bind an NPT protocol to a driver .....	895
<code>muxTkCookieGet()</code>	– returns the cookie for a device .....	897
<code>muxTkDrvCheck()</code>	– checks if the device is an NPT or an END interface .....	898
<code>muxTkPollReceive()</code>	– poll for a packet from a NPT or END driver .....	898
<code>muxTkPollSend()</code>	– send a packet out in polled mode to an END or NPT interface .....	899
<code>muxTkReceive()</code>	– receive a packet from a NPT driver .....	901
<code>muxTkSend()</code>	– send a packet out on a Toolkit or END network interface .....	902
<code>muxUnbind()</code>	– detach a network service from the specified device .....	904
<code>mv()</code>	– mv file into other directory. ....	905
<code>nanosleep()</code>	– suspend the current task until the time interval elapses (POSIX) .....	906
<code>netBufLibInit()</code>	– initialize <b>netBufLib</b> .....	907
<code>netCIBlkFree()</code>	– free a <code>clBlk</code> -cluster construct back to the memory pool .....	907
<code>netCIBlkGet()</code>	– get a <code>clBlk</code> .....	908
<code>netCIBlkJoin()</code>	– join a cluster to a <code>clBlk</code> structure .....	908
<code>netCIFree()</code>	– free a cluster back to the memory pool .....	909
<code>netCIPoolIdGet()</code>	– return a <code>CL_POOL_ID</code> for a specified buffer size .....	910
<code>netClusterGet()</code>	– get a cluster from the specified cluster pool .....	911
<code>netDevCreate()</code>	– create a remote file device .....	911
<code>netDevCreate2()</code>	– create a remote file device with fixed buffer size .....	912
<code>netDrv()</code>	– install the network remote file driver .....	913
<code>netDrvDebugLevelSet()</code>	– set the debug level of the <b>netDrv</b> library routines .....	913
<code>netDrvFileDoesNotExistInstall()</code>	– install an applet to test if a file exists .....	914
<code>netHelp()</code>	– print a synopsis of network routines .....	914
<code>netLibInit()</code>	– initialize the network package .....	915
<code>netMblkChainDup()</code>	– duplicate an <code>mBlk</code> chain .....	916
<code>netMblkCIChainFree()</code>	– free a chain of <code>mBlk-clBlk</code> -cluster constructs .....	917
<code>netMblkCIFree()</code>	– free an <code>mBlk-clBlk</code> -cluster construct .....	917
<code>netMblkCIGet()</code>	– get a <code>clBlk</code> -cluster and join it to the specified <code>mBlk</code> .....	918
<code>netMblkCIJoin()</code>	– join an <code>mBlk</code> to a <code>clBlk</code> -cluster construct .....	919
<code>netMblkDup()</code>	– duplicate an <code>mBlk</code> .....	920
<code>netMblkFree()</code>	– free an <code>mBlk</code> back to its memory pool .....	921
<code>netMblkGet()</code>	– get an <code>mBlk</code> from a memory pool .....	921
<code>netMblkToBufCopy()</code>	– copy data from an <code>mBlk</code> to a buffer .....	922
<code>netPoolDelete()</code>	– delete a memory pool .....	923
<code>netPoolInit()</code>	– initialize a <b>netBufLib</b> -managed memory pool .....	923
<code>netPoolKheapInit()</code>	– kernel heap version of <b>netPoolInit()</b> .....	927
<code>netPoolShow()</code>	– show pool statistics .....	927
<code>netShowInit()</code>	– initialize network show routines .....	928
<code>netStackDataPoolShow()</code>	– show network stack data pool statistics .....	929
<code>netStackSysPoolShow()</code>	– show network stack system pool statistics .....	929
<code>netTask()</code>	– network task entry point .....	930
<code>netTupleGet()</code>	– get an <code>mBlk-clBlk</code> -cluster .....	930
<code>nextIndex()</code>	– the comparison routine for the AVL tree .....	932

<b>nfsAuthUnixGet()</b>	– get the NFS UNIX authentication parameters .....	932
<b>nfsAuthUnixPrompt()</b>	– modify the NFS UNIX authentication parameters.....	933
<b>nfsAuthUnixSet()</b>	– set the NFS UNIX authentication parameters.....	933
<b>nfsAuthUnixShow()</b>	– display the NFS UNIX authentication parameters.....	934
<b>nfsDevInfoGet()</b>	– read configuration information from the requested NFS device .....	934
<b>nfsDevListGet()</b>	– create list of all the NFS devices in the system .....	935
<b>nfsDevShow()</b>	– display the mounted NFS devices .....	935
<b>nfsdInit()</b>	– initialize the NFS server .....	936
<b>nfsDrv()</b>	– install the NFS driver.....	937
<b>nfsDrvNumGet()</b>	– return the IO system driver number for the NFS driver .....	937
<b>nfsdStatusGet()</b>	– get the status of the NFS server.....	938
<b>nfsdStatusShow()</b>	– show the status of the NFS server.....	938
<b>nfsExport()</b>	– specify a file system to be NFS exported .....	939
<b>nfsExportShow()</b>	– display the exported file systems of a remote host .....	939
<b>nfsHelp()</b>	– display the NFS help menu .....	940
<b>nfsIdSet()</b>	– set the ID number of the NFS UNIX authentication parameters .....	941
<b>nfsMount()</b>	– mount an NFS file system .....	941
<b>nfsMountAll()</b>	– mount all file systems exported by a specified host .....	942
<b>nfsUnexport()</b>	– remove a file system from the list of exported file systems.....	942
<b>nfsUnmount()</b>	– unmount an NFS device.....	943
<b>ntPassFsDevInit()</b>	– associate a device with ntPassFs file system functions.....	943
<b>ntPassFsInit()</b>	– prepare to use the ntPassFs library.....	944
<b>open()</b>	– open a file .....	945
<b>opendir()</b>	– open a directory for searching (POSIX) .....	946
<b>operator delete()</b>	– default run-time support for memory deallocation (C++).....	946
<b>operator new()</b>	– default run-time support for operator new (C++) .....	947
<b>operator new()</b>	– default run-time support for operator new (nothrow) (C++) .....	947
<b>operator new()</b>	– run-time support for operator new with placement (C++).....	948
<b>passFsDevInit()</b>	– associate a device with passFs file system functions .....	949
<b>passFsInit()</b>	– prepare to use the passFs library .....	949
<b>pause()</b>	– suspend the task until delivery of a signal (POSIX).....	950
<b>pc()</b>	– return the contents of the program counter .....	950
<b>pentiumBtc()</b>	– execute atomic compare-and-exchange instruction to clear a bit .....	951
<b>pentiumBts()</b>	– execute atomic compare-and-exchange instruction to set a bit.....	951
<b>pentiumCr4Get()</b>	– get contents of CR4 register .....	952
<b>pentiumCr4Set()</b>	– sets specified value to the CR4 register.....	952
<b>pentiumMcaEnable()</b>	– enable/disable the MCA (Machine Check Architecture).....	953
<b>pentiumMcaShow()</b>	– show MCA (Machine Check Architecture) registers .....	953
<b>pentiumMsrGet()</b>	– get the contents of the specified MSR (Model Specific Register) .....	954
<b>pentiumMsrInit()</b>	– initialize all the MSRs (Model Specific Register) .....	954
<b>pentiumMsrSet()</b>	– set a value to the specified MSR (Model Specific Registers).....	955
<b>pentiumMsrShow()</b>	– show all the MSR (Model Specific Register) .....	955
<b>pentiumMtrrDisable()</b>	– disable MTRR (Memory Type Range Register) .....	956
<b>pentiumMtrrEnable()</b>	– enable MTRR (Memory Type Range Register) .....	956
<b>pentiumMtrrGet()</b>	– get MTRRs to a specified MTRR table.....	957

<code>pentiumMtrrSet()</code>	– set MTRRs from specified MTRR table with WRMSR instruction .....	957
<code>pentiumP5PmcGet()</code>	– get the contents of P5 PMC0 and PMC1 .....	958
<code>pentiumP5PmcGet0()</code>	– get the contents of P5 PMC0 .....	958
<code>pentiumP5PmcGet1()</code>	– get the contents of P5 PMC1 .....	959
<code>pentiumP5PmcReset()</code>	– reset both PMC0 and PMC1 .....	959
<code>pentiumP5PmcReset0()</code>	– reset PMC0 .....	960
<code>pentiumP5PmcReset1()</code>	– reset PMC1 .....	960
<code>pentiumP5PmcStart0()</code>	– start PMC0 .....	961
<code>pentiumP5PmcStart1()</code>	– start PMC1 .....	961
<code>pentiumP5PmcStop()</code>	– stop both P5 PMC0 and PMC1 .....	962
<code>pentiumP5PmcStop0()</code>	– stop P5 PMC0 .....	962
<code>pentiumP5PmcStop1()</code>	– stop P5 PMC1 .....	963
<code>pentiumP6PmcGet()</code>	– get the contents of PMC0 and PMC1 .....	963
<code>pentiumP6PmcGet0()</code>	– get the contents of PMC0 .....	964
<code>pentiumP6PmcGet1()</code>	– get the contents of PMC1 .....	964
<code>pentiumP6PmcReset()</code>	– reset both PMC0 and PMC1 .....	965
<code>pentiumP6PmcReset0()</code>	– reset PMC0 .....	965
<code>pentiumP6PmcReset1()</code>	– reset PMC1 .....	965
<code>pentiumP6PmcStart()</code>	– start both PMC0 and PMC1 .....	966
<code>pentiumP6PmcStop()</code>	– stop both PMC0 and PMC1 .....	966
<code>pentiumP6PmcStop1()</code>	– stop PMC1 .....	967
<code>pentiumPmcGet()</code>	– get the contents of PMC0 and PMC1 .....	967
<code>pentiumPmcGet0()</code>	– get the contents of PMC0 .....	968
<code>pentiumPmcGet1()</code>	– get the contents of PMC1 .....	968
<code>pentiumPmcReset()</code>	– reset both PMC0 and PMC1 .....	969
<code>pentiumPmcReset0()</code>	– reset PMC0 .....	969
<code>pentiumPmcReset1()</code>	– reset PMC1 .....	969
<code>pentiumPmcShow()</code>	– show PMCs (Performance Monitoring Counters).....	970
<code>pentiumPmcStart()</code>	– start both PMC0 and PMC1 .....	970
<code>pentiumPmcStart0()</code>	– start PMC0 .....	971
<code>pentiumPmcStart1()</code>	– start PMC1 .....	971
<code>pentiumPmcStop()</code>	– stop both PMC0 and PMC1 .....	972
<code>pentiumPmcStop0()</code>	– stop PMC0 .....	972
<code>pentiumPmcStop1()</code>	– stop PMC1 .....	972
<code>pentiumSerialize()</code>	– execute a serializing instruction CPUID .....	973
<code>pentiumTlbFlush()</code>	– flush TLBs (Translation Lookaside Buffers) .....	973
<code>pentiumTscGet32()</code>	– get the lower half of the 64Bit TSC (Timestamp Counter) .....	974
<code>pentiumTscGet64()</code>	– get 64Bit TSC (Timestamp Counter) .....	974
<code>pentiumTscReset()</code>	– reset the TSC (Timestamp Counter) .....	974
<code>period()</code>	– spawn a task to call a function periodically .....	975
<code>periodRun()</code>	– call a function periodically .....	976
<code>perror()</code>	– map an error number in <code>errno</code> to an error message (ANSI) .....	976
<code>ping()</code>	– test that a remote host is reachable .....	977
<code>pingLibInit()</code>	– initialize the <code>ping()</code> utility .....	978
<code>pipeDevCreate()</code>	– create a pipe device .....	979

<code>pipeDevDelete()</code>	– delete a pipe device .....	979
<code>pipeDrv()</code>	– initialize the pipe driver .....	980
<code>pow()</code>	– compute the value of a number raised to a specified power (ANSI).....	981
<code>powf()</code>	– compute the value of a number raised to a specified power (ANSI).....	982
<code>pppDelete()</code>	– delete a PPP network interface .....	982
<code>pppHookAdd()</code>	– add a hook routine on a unit basis .....	983
<code>pppHookDelete()</code>	– delete a hook routine on a unit basis .....	983
<code>pppInfoGet()</code>	– get PPP link status information .....	984
<code>pppInfoShow()</code>	– display PPP link status information .....	985
<code>pppInit()</code>	– initialize a PPP network interface .....	985
<code>pppSecretAdd()</code>	– add a secret to the PPP authentication secrets table.....	993
<code>pppSecretDelete()</code>	– delete a secret from the PPP authentication secrets table .....	994
<code>pppSecretShow()</code>	– display the PPP authentication secrets table .....	994
<code>pppstatGet()</code>	– get PPP link statistics .....	995
<code>pppstatShow()</code>	– display PPP link statistics.....	995
<code>printErr()</code>	– write a formatted string to the standard error stream .....	996
<code>printErrno()</code>	– print the definition of a specified error status value .....	996
<code>printf()</code>	– write a formatted string to the standard output stream (ANSI) .....	997
<code>printLogo()</code>	– print the VxWorks logo .....	1001
<code>proxyArpLibInit()</code>	– initialize proxy ARP .....	1001
<code>proxyNetCreate()</code>	– create a proxy ARP network .....	1002
<code>proxyNetDelete()</code>	– delete a proxy network .....	1002
<code>proxyNetShow()</code>	– show proxy ARP networks .....	1003
<code>proxyPortFwdOff()</code>	– disable broadcast forwarding for a particular port .....	1003
<code>proxyPortFwdOn()</code>	– enable broadcast forwarding for a particular port .....	1004
<code>proxyPortShow()</code>	– show ports enabled for broadcast forwarding .....	1004
<code>proxyReg()</code>	– register a proxy client.....	1005
<code>proxyUnreg()</code>	– unregister a proxy client.....	1005
<code>psrShow()</code>	– display the meaning of a specified psr value, symbolically (ARM) .....	1006
<code>pthreadLibInit()</code>	– initialize POSIX threads support.....	1007
<code>pthread_attr_destroy()</code>	– destroy a thread attributes object (POSIX).....	1007
<code>pthread_attr_getdetachstate()</code>	– get value of detachstate attribute in thread attributes object (POSIX) ..	1008
<code>pthread_attr_getinheritsched()</code>	– get value of inheritsched attribute in thread attributes object (POSIX)	1008
<code>pthread_attr_getname()</code>	– get name of thread attribute object .....	1009
<code>pthread_attr_getschedparam()</code>	– get value of schedparam attribute in thread attributes object (POSIX)	1009
<code>pthread_attr_getschedpolicy()</code>	– get schedpolicy attribute from thread attributes object (POSIX).....	1010
<code>pthread_attr_getscope()</code>	– get contention scope from thread attributes (POSIX) .....	1011
<code>pthread_attr_getstackaddr()</code>	– get value of stackaddr attribute from thread attributes object (POSIX) ..	1011
<code>pthread_attr_getstacksize()</code>	– get stack value of stacksize attribute in thread attributes object (POSIX)	1012
<code>pthread_attr_init()</code>	– initialize thread attributes object (POSIX).....	1012
<code>pthread_attr_setdetachstate()</code>	– set detachstate attribute in thread attributes object (POSIX) .....	1013
<code>pthread_attr_setinheritsched()</code>	– set inheritsched attribute in thread attribute object (POSIX).....	1014
<code>pthread_attr_setname()</code>	– set name in thread attribute object .....	1015
<code>pthread_attr_setschedparam()</code>	– set schedparam attribute in thread attributes object (POSIX) .....	1015
<code>pthread_attr_setschedpolicy()</code>	– set schedpolicy attribute in thread attributes object (POSIX) .....	1016

<code>pthread_attr_setscope()</code>	– set contention scope for thread attributes (POSIX) .....	1017
<code>pthread_attr_setstackaddr()</code>	– set stackaddr attribute in thread attributes object (POSIX) .....	1017
<code>pthread_attr_setstacksize()</code>	– set stacksize attribute in thread attributes object (POSIX) .....	1018
<code>pthread_cancel()</code>	– cancel execution of a thread (POSIX) .....	1018
<code>pthread_cleanup_pop()</code>	– pop a cleanup routine off the top of the stack (POSIX) .....	1019
<code>pthread_cleanup_push()</code>	– pushes a routine onto the cleanup stack (POSIX) .....	1019
<code>pthread_cond_broadcast()</code>	– unblock all threads waiting on a condition (POSIX) .....	1020
<code>pthread_cond_destroy()</code>	– destroy a condition variable (POSIX) .....	1020
<code>pthread_cond_init()</code>	– initialize condition variable (POSIX) .....	1021
<code>pthread_cond_signal()</code>	– unblock a thread waiting on a condition (POSIX) .....	1022
<code>pthread_cond_timedwait()</code>	– wait for a condition variable with a timeout (POSIX) .....	1022
<code>pthread_cond_wait()</code>	– wait for a condition variable (POSIX) .....	1023
<code>pthread_condattr_destroy()</code>	– destroy a condition attributes object (POSIX) .....	1024
<code>pthread_condattr_init()</code>	– initialize a condition attribute object (POSIX) .....	1024
<code>pthread_create()</code>	– create a thread (POSIX) .....	1025
<code>pthread_detach()</code>	– dynamically detach a thread (POSIX) .....	1025
<code>pthread_equal()</code>	– compare thread IDs (POSIX) .....	1026
<code>pthread_exit()</code>	– terminate a thread (POSIX) .....	1026
<code>pthread_getschedparam()</code>	– get value of schedparam attribute from a thread (POSIX) .....	1027
<code>pthread_getspecific()</code>	– get thread specific data (POSIX) .....	1027
<code>pthread_join()</code>	– wait for a thread to terminate (POSIX) .....	1028
<code>pthread_key_create()</code>	– create a thread specific data key (POSIX) .....	1029
<code>pthread_key_delete()</code>	– delete a thread specific data key (POSIX) .....	1029
<code>pthread_kill()</code>	– send a signal to a thread (POSIX) .....	1030
<code>pthread_mutex_destroy()</code>	– destroy a mutex (POSIX) .....	1030
<code>pthread_mutex_getprioceiling()</code>	– get the value of the prioceiling attribute of a mutex (POSIX) ..	1031
<code>pthread_mutex_init()</code>	– initialize mutex from attributes object (POSIX) .....	1031
<code>pthread_mutex_lock()</code>	– lock a mutex (POSIX) .....	1032
<code>pthread_mutex_setprioceiling()</code>	– dynamically set the prioceiling attribute of a mutex (POSIX) ..	1033
<code>pthread_mutex_trylock()</code>	– lock mutex if it is available (POSIX) .....	1033
<code>pthread_mutex_unlock()</code>	– unlock a mutex (POSIX) .....	1034
<code>pthread_mutexattr_destroy()</code>	– destroy mutex attributes object (POSIX) .....	1034
<code>pthread_mutexattr_getprioceiling()</code>	– get value of prioceiling attr in a mutex attr object (POSIX) .....	1035
<code>pthread_mutexattr_getprotocol()</code>	– get value of protocol in mutex attributes object (POSIX) .....	1036
<code>pthread_mutexattr_init()</code>	– initialize mutex attributes object (POSIX) .....	1036
<code>pthread_mutexattr_setprioceiling()</code>	– set prioceiling attr in mutex attributes object (POSIX) .....	1037
<code>pthread_mutexattr_setprotocol()</code>	– set protocol attribute in mutex attribute object (POSIX) .....	1037
<code>pthread_once()</code>	– dynamic package initialization (POSIX) .....	1038
<code>pthread_self()</code>	– get the calling thread's ID (POSIX) .....	1039
<code>pthread_setcancelstate()</code>	– set cancellation state for calling thread (POSIX) .....	1039
<code>pthread_setcanceltype()</code>	– set cancellation type for calling thread (POSIX) .....	1040
<code>pthread_setschedparam()</code>	– dynamically set schedparam attribute for a thread (POSIX) ....	1040
<code>pthread_setspecific()</code>	– set thread specific data (POSIX) .....	1041
<code>pthread_sigmask()</code>	– change and/or examine calling thread's signal mask (POSIX)	1042
<code>pthread_testcancel()</code>	– create a cancellation point in the calling thread (POSIX) .....	1043

<b>ptyDevCreate()</b>	– create a pseudo terminal .....	1043
<b>ptyDevRemove()</b>	– destroy a pseudo terminal .....	1044
<b>ptyDrv()</b>	– initialize the pseudo-terminal driver .....	1044
<b>ptyShow()</b>	– show the state of the Pty Buffers.....	1045
<b>putc()</b>	– write a character to a stream (ANSI) .....	1045
<b>putchar()</b>	– write a character to the standard output stream (ANSI).....	1046
<b>putenv()</b>	– set an environment variable .....	1046
<b>puts()</b>	– write a string to the standard output stream (ANSI).....	1047
<b>putw()</b>	– write a word (32-bit integer) to a stream .....	1047
<b>pwd()</b>	– print the current default directory.....	1048
<b>qsort()</b>	– sort an array of objects (ANSI) .....	1049
<b>r0()</b>	– return the contents of register r0 (also r1 - r14, r1-r15 for SH) (ARM, SH).....	1050
<b>raise()</b>	– send a signal to the caller’s task .....	1050
<b>ramDevCreate()</b>	– create a RAM disk device.....	1051
<b>ramDiskDevCreate()</b>	– initialize a RAM Disk device .....	1052
<b>ramDrv()</b>	– prepare a RAM disk driver for use (optional) .....	1053
<b>rand()</b>	– generate a pseudo-random integer between 0 and <b>RAND_MAX</b> (ANSI)....	1054
<b>rawFsDevInit()</b>	– associate a block device with raw volume functions.....	1054
<b>rawFsInit()</b>	– prepare to use the raw volume library.....	1055
<b>rawFsModeChange()</b>	– modify the mode of a raw device volume.....	1055
<b>rawFsReadyChange()</b>	– notify <b>rawFsLib</b> of a change in ready status.....	1056
<b>rawFsVolUnmount()</b>	– disable a raw device volume .....	1056
<b>rcmd()</b>	– execute a shell command on a remote machine .....	1057
<b>rcvEtherAddrAdd()</b>	– add a physical address into the linked list .....	1058
<b>rcvEtherAddrGet()</b>	– populate the rcvAddr fields for the ifRcvAddressTable .....	1058
<b>rdCtl()</b>	– implement the ICMP router discovery control function.....	1059
<b>rdisc()</b>	– implement the ICMP router discovery function .....	1061
<b>rdiscIfReset()</b>	– check for new or removed interfaces for router discovery .....	1062
<b>rdiscInit()</b>	– initialize the ICMP router discovery function .....	1062
<b>rdiscLibInit()</b>	– initialize router discovery .....	1063
<b>rdiscTimerEvent()</b>	– called after watchdog timeout.....	1063
<b>read()</b>	– read bytes from a file or device .....	1064
<b>readdir()</b>	– read one entry from a directory (POSIX).....	1064
<b>realloc()</b>	– reallocate a block of memory (ANSI) .....	1065
<b>reboot()</b>	– reset network devices and transfer control to boot ROMs.....	1066
<b>rebootHookAdd()</b>	– add a routine to be called at reboot .....	1067
<b>recv()</b>	– receive data from a socket.....	1067
<b>recvfrom()</b>	– receive a message from a socket .....	1068
<b>recvmsg()</b>	– receive a message from a socket .....	1069
<b>reld()</b>	– reload an object module .....	1069
<b>remCurIdGet()</b>	– get the current user name and password .....	1070
<b>remCurIdSet()</b>	– set the remote user name and password .....	1070
<b>remove()</b>	– remove a file (ANSI) .....	1071
<b>rename()</b>	– change the name of a file.....	1072
<b>repeat()</b>	– spawn a task to call a function repeatedly .....	1072

<code>repeatRun()</code>	– call a function repeatedly .....	1073
<code>resolvDNComp()</code>	– compress a DNS name in a DNS packet .....	1074
<code>resolvDNExpand()</code>	– expand a DNS compressed name from a DNS packet.....	1074
<code>resolvGetHostByAddr()</code>	– query the DNS server for the host name of an IP address .....	1075
<code>resolvGetHostByName()</code>	– query the DNS server for the IP address of a host .....	1076
<code>resolvInit()</code>	– initialize the resolver library .....	1077
<code>resolvMkQuery()</code>	– create all types of DNS queries.....	1078
<code>resolvParamsGet()</code>	– get the parameters which control the resolver library .....	1078
<code>resolvParamsSet()</code>	– set the parameters which control the resolver library .....	1079
<code>resolvQuery()</code>	– construct a query, send it, wait for a response.....	1080
<code>resolvSend()</code>	– send a pre-formatted query and return the answer .....	1081
<code>rewind()</code>	– set the file position indicator to the beginning of a file (ANSI).....	1082
<code>rewinddir()</code>	– reset position to the start of a directory (POSIX).....	1082
<code>rindex()</code>	– find the last occurrence of a character in a string .....	1083
<code>ripAddrsXtract()</code>	– extract socket address pointers from the route message.....	1083
<code>ripAuthHook()</code>	– sample authentication hook.....	1084
<code>ripAuthHookAdd()</code>	– add an authentication hook to a RIP interface .....	1085
<code>ripAuthHookDelete()</code>	– remove an authentication hook from a RIP interface .....	1087
<code>ripAuthKeyAdd()</code>	– add a new RIP authentication key .....	1088
<code>ripAuthKeyDelete()</code>	– delete an existing RIP authentication key.....	1088
<code>ripAuthKeyFind()</code>	– find a RIP authentication key .....	1089
<code>ripAuthKeyFindFirst()</code>	– find a RIP authentication key .....	1089
<code>ripAuthKeyInMD5()</code>	– authenticate an incoming RIP-2 message using MD5.....	1090
<code>ripAuthKeyOut1MD5()</code>	– start MD5 authentication of an outgoing RIP-2 message .....	1090
<code>ripAuthKeyOut2MD5()</code>	– authenticate an outgoing RIP-2 message using MD5 .....	1091
<code>ripAuthKeyShow()</code>	– show current authentication configuration .....	1091
<code>ripDebugLevelSet()</code>	– specify amount of debugging output .....	1092
<code>ripFilterDisable()</code>	– prevent strict border gateway filtering .....	1092
<code>ripFilterEnable()</code>	– activate strict border gateway filtering .....	1093
<code>ripIfExcludeListAdd()</code>	– add an interface to the RIP exclusion list.....	1093
<code>ripIfExcludeListDelete()</code>	– delete an interface from RIP exclusion list.....	1094
<code>ripIfExcludeListShow()</code>	– show the RIP interface exclusion list.....	1094
<code>ripIfReset()</code>	– alter the RIP configuration after an interface changes .....	1095
<code>ripIfSearch()</code>	– add new interfaces to the internal list .....	1095
<code>ripIfShow()</code>	– display the internal interface table maintained by RIP.....	1096
<code>ripLeakHookAdd()</code>	– add a hook to bypass the RIP and kernel routing tables .....	1096
<code>ripLeakHookDelete()</code>	– remove a table bypass hook from a RIP interface .....	1097
<code>ripLibInit()</code>	– initialize the RIP routing library .....	1097
<code>ripRouteHookAdd()</code>	– add a hook to install static and non-RIP routes into RIP.....	1099
<code>ripRouteHookDelete()</code>	– remove the route hook.....	1102
<code>ripRouteShow()</code>	– display the internal routing table maintained by RIP.....	1102
<code>ripSendHookAdd()</code>	– add an update filter to a RIP interface.....	1103
<code>ripSendHookDelete()</code>	– remove an update filter from a RIP interface .....	1104
<code>ripShutdown()</code>	– terminate all RIP processing .....	1104
<code>rlogin()</code>	– log in to a remote host .....	1105

<code>rlogind()</code>	– the VxWorks remote login daemon .....	1105
<code>rlogInit()</code>	– initialize the remote login facility .....	1106
<code>rm()</code>	– remove a file.....	1107
<code>rmdir()</code>	– remove a directory .....	1107
<code>rngBufGet()</code>	– get characters from a ring buffer .....	1108
<code>rngBufPut()</code>	– put bytes into a ring buffer .....	1108
<code>rngCreate()</code>	– create an empty ring buffer .....	1109
<code>rngDelete()</code>	– delete a ring buffer.....	1109
<code>rngFlush()</code>	– make a ring buffer empty .....	1110
<code>rngFreeBytes()</code>	– determine the number of free bytes in a ring buffer.....	1110
<code>rngIsEmpty()</code>	– test if a ring buffer is empty .....	1111
<code>rngIsFull()</code>	– test if a ring buffer is full (no more room).....	1111
<code>rngMoveAhead()</code>	– advance a ring pointer by n bytes .....	1112
<code>rngNBytes()</code>	– determine the number of bytes in a ring buffer .....	1112
<code>rngPutAhead()</code>	– put a byte ahead in a ring buffer without moving ring pointers.....	1113
<code>romStart()</code>	– generic ROM initialization.....	1113
<code>round()</code>	– round a number to the nearest integer .....	1114
<code>roundf()</code>	– round a number to the nearest integer .....	1114
<code>routeAdd()</code>	– add a route .....	1115
<code>routeDelete()</code>	– delete a route .....	1116
<code>routeEntryAdd()</code>	– insert a route in the routing table .....	1116
<code>routeEntryDel()</code>	– remove a route from the routing table.....	1117
<code>routeEntryLookup()</code>	– find a matching route for a destination .....	1118
<code>routeModify()</code>	– change an entry in the routing table .....	1119
<code>routeNetAdd()</code>	– add a route to a destination that is a network .....	1120
<code>routeShow()</code>	– display all IP routes (summary information) .....	1120
<code>routeStatShow()</code>	– display routing statistics .....	1122
<code>routeStorageUnbind()</code>	– remove a registered handler from the routing system.....	1122
<code>routeTableWalk()</code>	– traverse the IP routing table .....	1123
<code>rpcInit()</code>	– initialize the RPC package .....	1124
<code>rpcTaskInit()</code>	– initialize a task’s access to the RPC package.....	1124
<code>rresvport()</code>	– open a socket with a privileged port bound to it .....	1125
<code>rt11FsDateSet()</code>	– set the rt11Fs file system date.....	1125
<code>rt11FsDevInit()</code>	– initialize the rt11Fs device descriptor .....	1126
<code>rt11FsInit()</code>	– prepare to use the rt11Fs library .....	1127
<code>rt11FsMkfs()</code>	– initialize a device and create an rt11Fs file system .....	1127
<code>rt11FsModeChange()</code>	– modify the mode of an rt11Fs volume.....	1128
<code>rt11FsReadyChange()</code>	– notify rt11Fs of a change in ready status .....	1128
<code>s()</code>	– single-step a task .....	1130
<code>scanf()</code>	– read and convert characters from the standard input stream (ANSI) .....	1130
<code>sched_get_priority_max()</code>	– get the maximum priority (POSIX) .....	1131
<code>sched_get_priority_min()</code>	– get the minimum priority (POSIX).....	1132
<code>sched_getparam()</code>	– get the scheduling parameters for a specified task (POSIX).....	1132
<code>sched_getscheduler()</code>	– get the current scheduling policy (POSIX).....	1133
<code>sched_rr_get_interval()</code>	– get the current time slice (POSIX).....	1134

<code>sched_setparam()</code>	– set a task’s priority (POSIX) .....	1134
<code>sched_setscheduler()</code>	– set scheduling policy and scheduling parameters (POSIX) .....	1135
<code>sched_yield()</code>	– relinquish the CPU (POSIX).....	1136
<code>scsi2IfInIt()</code>	– initialize the SCSI-2 interface to <b>scsiLib</b> .....	1136
<code>scsiAutoConfig()</code>	– configure all devices connected to a SCSI controller .....	1137
<code>scsiBlkDevCreate()</code>	– define a logical partition on a SCSI block device.....	1137
<code>scsiBlkDevInit()</code>	– initialize fields in a SCSI logical partition.....	1138
<code>scsiBlkDevShow()</code>	– show the <b>BLK_DEV</b> structures on a specified physical device.....	1139
<code>scsiBusReset()</code>	– pulse the reset signal on the SCSI bus .....	1139
<code>scsiCacheSnoopDisable()</code>	– inform SCSI that hardware snooping of caches is disabled .....	1140
<code>scsiCacheSnoopEnable()</code>	– inform SCSI that hardware snooping of caches is enabled .....	1140
<code>scsiCacheSynchronize()</code>	– synchronize the caches for data coherency .....	1141
<code>scsiErase()</code>	– issue an ERASE command to a SCSI device.....	1142
<code>scsiFormatUnit()</code>	– issue a <b>FORMAT_UNIT</b> command to a SCSI device .....	1142
<code>scsiIdentMsgBuild()</code>	– build an identification message.....	1143
<code>scsiIdentMsgParse()</code>	– parse an identification message .....	1143
<code>scsiInquiry()</code>	– issue an INQUIRY command to a SCSI device.....	1144
<code>scsiIoctl()</code>	– perform a device-specific I/O control function .....	1145
<code>scsiLoadUnit()</code>	– issue a <b>LOAD/UNLOAD</b> command to a SCSI device .....	1145
<code>scsiMgrBusReset()</code>	– handle a controller-bus reset event.....	1146
<code>scsiMgrCtrlEvent()</code>	– send an event to the SCSI controller state machine.....	1146
<code>scsiMgrEventNotify()</code>	– notify the SCSI manager of a SCSI (controller) event .....	1147
<code>scsiMgrShow()</code>	– show status information for the SCSI manager .....	1147
<code>scsiMgrThreadEvent()</code>	– send an event to the thread state machine.....	1148
<code>scsiModeSelect()</code>	– issue a <b>MODE_SELECT</b> command to a SCSI device.....	1149
<code>scsiModeSense()</code>	– issue a <b>MODE_SENSE</b> command to a SCSI device.....	1149
<code>scsiMsgInComplete()</code>	– handle a complete SCSI message received from the target.....	1150
<code>scsiMsgOutComplete()</code>	– perform post-processing after a SCSI message is sent.....	1150
<code>scsiMsgOutReject()</code>	– perform post-processing when an outgoing message is rejected.....	1151
<code>scsiPhysDevCreate()</code>	– create a SCSI physical device structure.....	1151
<code>scsiPhysDevDelete()</code>	– delete a SCSI physical-device structure .....	1152
<code>scsiPhysDevIdGet()</code>	– return a pointer to a <b>SCSI_PHYS_DEV</b> structure .....	1152
<code>scsiPhysDevShow()</code>	– show status information for a physical device.....	1153
<code>scsiRdSecs()</code>	– read sector(s) from a SCSI block device .....	1154
<code>scsiRdTape()</code>	– read bytes or blocks from a SCSI tape device .....	1154
<code>scsiReadCapacity()</code>	– issue a <b>READ_CAPACITY</b> command to a SCSI device.....	1155
<code>scsiRelease()</code>	– issue a <b>RELEASE</b> command to a SCSI device.....	1155
<code>scsiReleaseUnit()</code>	– issue a <b>RELEASE UNIT</b> command to a SCSI device .....	1156
<code>scsiReqSense()</code>	– issue a <b>REQUEST_SENSE</b> command to a SCSI device and read results	1156
<code>scsiReserve()</code>	– issue a <b>RESERVE</b> command to a SCSI device .....	1157
<code>scsiReserveUnit()</code>	– issue a <b>RESERVE UNIT</b> command to a SCSI device .....	1157
<code>scsiRewind()</code>	– issue a <b>REWIND</b> command to a SCSI device .....	1158
<code>scsiSeqDevCreate()</code>	– create a SCSI sequential device .....	1158
<code>scsiSeqIoctl()</code>	– perform an I/O control function for sequential access devices.....	1159
<code>scsiSeqReadBlockLimits()</code>	– issue a <b>READ_BLOCK_LIMITS</b> command to a SCSI device.....	1159

<code>scsiSeqStatusCheck()</code>	– detect a change in media.....	1160
<code>scsiShow()</code>	– list the physical devices attached to a SCSI controller.....	1160
<code>scsiSpace()</code>	– move the tape on a specified physical SCSI device .....	1161
<code>scsiStartStopUnit()</code>	– issue a <code>START_STOP_UNIT</code> command to a SCSI device.....	1162
<code>scsiSyncXferNegotiate()</code>	– initiate or continue negotiating transfer parameters .....	1162
<code>scsiTapeModeSelect()</code>	– issue a <code>MODE_SELECT</code> command to a SCSI tape device.....	1163
<code>scsiTapeModeSense()</code>	– issue a <code>MODE_SENSE</code> command to a SCSI tape device.....	1163
<code>scsiTargetOptionsGet()</code>	– get options for one or all SCSI targets.....	1164
<code>scsiTargetOptionsSet()</code>	– set options for one or all SCSI targets .....	1164
<code>scsiTargetOptionsShow()</code>	– display options for specified SCSI target.....	1165
<code>scsiTestUnitRdy()</code>	– issue a <code>TEST_UNIT_READY</code> command to a SCSI device.....	1166
<code>scsiThreadInit()</code>	– perform generic SCSI thread initialization.....	1166
<code>scsiWideXferNegotiate()</code>	– initiate or continue negotiating wide parameters .....	1167
<code>scsiWrtFileMarks()</code>	– write file marks to a SCSI sequential device .....	1167
<code>scsiWrtSecs()</code>	– write sector(s) to a SCSI block device .....	1168
<code>scsiWrtTape()</code>	– write data to a SCSI tape device .....	1168
<code>select()</code>	– pend on a set of file descriptors .....	1169
<code>selectInit()</code>	– initialize the select facility.....	1170
<code>selNodeAdd()</code>	– add a wake-up node to a <code>select()</code> wake-up list.....	1171
<code>selNodeDelete()</code>	– find and delete a node from a <code>select()</code> wake-up list .....	1171
<code>selWakeup()</code>	– wake up a task pending in <code>select()</code> .....	1172
<code>selWakeupAll()</code>	– wake up all tasks in a <code>select()</code> wake-up list.....	1172
<code>selWakeupListInit()</code>	– initialize a <code>select()</code> wake-up list .....	1173
<code>selWakeupListLen()</code>	– get the number of nodes in a <code>select()</code> wake-up list .....	1173
<code>selWakeupListTerm()</code>	– terminate a <code>select()</code> wake-up list.....	1174
<code>selWakeupType()</code>	– get the type of a <code>select()</code> wake-up node.....	1174
<code>semBCreate()</code>	– create and initialize a binary semaphore.....	1175
<code>semBSmCreate()</code>	– create and initialize a shared memory binary semaphore (VxMP).....	1175
<code>semCCreate()</code>	– create and initialize a counting semaphore.....	1176
<code>semClear()</code>	– take a release 4.x semaphore, if the semaphore is available .....	1177
<code>semCreate()</code>	– create and initialize a release 4.x binary semaphore.....	1177
<code>semCSmCreate()</code>	– create and initialize shared memory counting semaphore (VxMP) .....	1178
<code>semDelete()</code>	– delete a semaphore .....	1179
<code>semEvStart()</code>	– start event notification process for a semaphore.....	1179
<code>semEvStop()</code>	– stop event notification process for a semaphore .....	1181
<code>semFlush()</code>	– unblock every task pending on a semaphore.....	1181
<code>semGive()</code>	– give a semaphore .....	1182
<code>semInfo()</code>	– get a list of task IDs that are blocked on a semaphore .....	1183
<code>semInit()</code>	– initialize a static binary semaphore.....	1183
<code>semMCreate()</code>	– create and initialize a mutual-exclusion semaphore .....	1184
<code>semMGiveForce()</code>	– give a mutual-exclusion semaphore without restrictions.....	1185
<code>semPxLibInit()</code>	– initialize POSIX semaphore support.....	1185
<code>semPxShowInit()</code>	– initialize the POSIX semaphore show facility.....	1186
<code>semShow()</code>	– show information about a semaphore .....	1186
<code>semShowInit()</code>	– initialize the semaphore show facility .....	1187

<b>semTake()</b>	– take a semaphore.....	1188
<b>sem_close()</b>	– close a named semaphore (POSIX).....	1188
<b>sem_destroy()</b>	– destroy an unnamed semaphore (POSIX).....	1189
<b>sem_getvalue()</b>	– get the value of a semaphore (POSIX).....	1190
<b>sem_init()</b>	– initialize an unnamed semaphore (POSIX).....	1191
<b>sem_open()</b>	– initialize/open a named semaphore (POSIX).....	1191
<b>sem_post()</b>	– unlock (give) a semaphore (POSIX).....	1193
<b>sem_trywait()</b>	– lock (take) a semaphore, returning error if unavailable (POSIX).....	1194
<b>sem_unlink()</b>	– remove a named semaphore (POSIX).....	1194
<b>sem_wait()</b>	– lock (take) a semaphore, blocking if not available (POSIX).....	1195
<b>send()</b>	– send data to a socket.....	1196
<b>sendAdvert()</b>	– send an advertisement to one location.....	1196
<b>sendAdvertAll()</b>	– send an advertisement to all active locations.....	1197
<b>sendmsg()</b>	– send a message to a socket.....	1197
<b>sendto()</b>	– send a message to a socket.....	1198
<b>set_new_handler()</b>	– set new_handler to user-defined function (C++).....	1199
<b>set_terminate()</b>	– set terminate to user-defined function (C++).....	1199
<b>setbuf()</b>	– specify the buffering for a stream (ANSI).....	1200
<b>setbuffer()</b>	– specify buffering for a stream.....	1200
<b>sethostname()</b>	– set the symbolic name of this machine.....	1201
<b>setjmp()</b>	– save the calling environment in a jmp_buf argument (ANSI).....	1201
<b>setlinebuf()</b>	– set line buffering for standard output or standard error.....	1202
<b>setlocale()</b>	– set the appropriate locale (ANSI).....	1203
<b>setsockopt()</b>	– set socket options.....	1203
<b>setvbuf()</b>	– specify buffering for a stream (ANSI).....	1211
<b>shell()</b>	– the shell entry point.....	1212
<b>shellHistory()</b>	– display or set the size of shell history.....	1212
<b>shellInit()</b>	– start the shell.....	1213
<b>shellLock()</b>	– lock access to the shell.....	1213
<b>shellOrigStdSet()</b>	– set the shell’s default input/output/error file descriptors.....	1214
<b>shellPromptSet()</b>	– change the shell prompt.....	1214
<b>shellScriptAbort()</b>	– signal the shell to stop processing a script.....	1215
<b>show()</b>	– print information on a specified object.....	1215
<b>shutdown()</b>	– shut down a network connection.....	1216
<b>sigaction()</b>	– examine and/or specify the action associated with a signal (POSIX).....	1216
<b>sigaddset()</b>	– add a signal to a signal set (POSIX).....	1217
<b>sigblock()</b>	– add to a set of blocked signals.....	1217
<b>sigdelset()</b>	– delete a signal from a signal set (POSIX).....	1218
<b>sigemptyset()</b>	– initialize a signal set with no signals included (POSIX).....	1218
<b>sigfillset()</b>	– initialize a signal set with all signals included (POSIX).....	1219
<b>sigInit()</b>	– initialize the signal facilities.....	1219
<b>sigismember()</b>	– test to see if a signal is in a signal set (POSIX).....	1220
<b>signal()</b>	– specify the handler associated with a signal.....	1220
<b>sigpending()</b>	– retrieve the set of pending signals blocked from delivery (POSIX).....	1221
<b>sigprocmask()</b>	– examine and/or change the signal mask (POSIX).....	1221

<b>sigqueue()</b>	– send a queued signal to a task .....	1222
<b>sigqueueInit()</b>	– initialize the queued signal facilities .....	1223
<b>sigsetmask()</b>	– set the signal mask .....	1223
<b>sigsuspend()</b>	– suspend the task until delivery of a signal (POSIX) .....	1224
<b>sigtimedwait()</b>	– wait for a signal .....	1224
<b>sigvec()</b>	– install a signal handler .....	1226
<b>sigwait()</b>	– wait for a signal to be delivered (POSIX) .....	1226
<b>sigwaitinfo()</b>	– wait for real-time signals .....	1227
<b>sin()</b>	– compute a sine (ANSI) .....	1228
<b>sincos()</b>	– compute both a sine and cosine .....	1228
<b>sincosf()</b>	– compute both a sine and cosine .....	1229
<b>sinf()</b>	– compute a sine (ANSI) .....	1229
<b>sinh()</b>	– compute a hyperbolic sine (ANSI) .....	1230
<b>sinhf()</b>	– compute a hyperbolic sine (ANSI) .....	1230
<b>sleep()</b>	– delay for a specified amount of time .....	1231
<b>smMemAddToPool()</b>	– add memory to shared memory system partition (VxMP) .....	1231
<b>smMemCalloc()</b>	– allocate memory for array from shared memory system partition (VxMP) ..	1232
<b>smMemFindMax()</b>	– find largest free block in shared memory system partition (VxMP) .....	1233
<b>smMemFree()</b>	– free a shared memory system partition block of memory (VxMP) .....	1233
<b>smMemMalloc()</b>	– allocate block of memory from shared memory system partition (VxMP) ...	1234
<b>smMemOptionsSet()</b>	– set debug options for shared memory system partition (VxMP) .....	1234
<b>smMemRealloc()</b>	– reallocate block of memory from shared memory system partition (VxMP)	1235
<b>smMemShow()</b>	– show the shared memory system partition blocks and statistics (VxMP) .....	1236
<b>smNameAdd()</b>	– add a name to the shared memory name database (VxMP) .....	1237
<b>smNameFind()</b>	– look up a shared memory object by name (VxMP) .....	1238
<b>smNameFindByValue()</b>	– look up a shared memory object by value (VxMP) .....	1239
<b>smNameRemove()</b>	– remove an object from the shared memory objects name database (VxMP).	1240
<b>smNameShow()</b>	– show the contents of the shared memory objects name database (VxMP) .....	1240
<b>smNetShow()</b>	– show information about a shared memory network .....	1241
<b>smObjAttach()</b>	– attach the calling CPU to the shared memory objects facility (VxMP) .....	1242
<b>smObjGlobalToLocal()</b>	– convert a global address to a local address (VxMP) .....	1243
<b>smObjInit()</b>	– initialize a shared memory objects descriptor (VxMP) .....	1244
<b>smObjLibInit()</b>	– install the shared memory objects facility (VxMP) .....	1245
<b>smObjLocalToGlobal()</b>	– convert a local address to a global address (VxMP) .....	1245
<b>smObjSetup()</b>	– initialize the shared memory objects facility (VxMP) .....	1246
<b>smObjShow()</b>	– display the current status of shared memory objects (VxMP) .....	1247
<b>smObjTimeoutLogEnable()</b>	– control logging of failed attempts to take a spin-lock (VxMP) .....	1248
<b>sntpcTimeGet()</b>	– retrieve the current time from a remote source .....	1249
<b>sntpsClockSet()</b>	– assign a routine to access the reference clock .....	1250
<b>sntpsConfigSet()</b>	– change SNTP server broadcast settings .....	1251
<b>sntpsNsecToFraction()</b>	– convert portions of a second to NTP format .....	1251
<b>so()</b>	– single-step, but step over a subroutine .....	1252
<b>socket()</b>	– open a socket .....	1252
<b>sockUploadPathClose()</b>	– close the socket upload path (Windview) .....	1253
<b>sockUploadPathCreate()</b>	– establish an upload path to the host using a socket (Windview) .....	1254

<b>sockUploadPathLibInit()</b>	– initialize <b>wvSockUploadPathLib</b> library (Windview) .....	1254
<b>sockUploadPathWrite()</b>	– write to the socket upload path (Windview).....	1255
<b>sp()</b>	– spawn a task with default parameters .....	1255
<b>sprintf()</b>	– write a formatted string to a buffer (ANSI) .....	1256
<b>spy()</b>	– begin periodic task activity reports.....	1257
<b>spyClkStart()</b>	– start collecting task activity data .....	1257
<b>spyClkStop()</b>	– stop collecting task activity data .....	1258
<b>spyHelp()</b>	– display task monitoring help menu.....	1258
<b>spyLibInit()</b>	– initialize task CPU utilization tool package.....	1259
<b>spyReport()</b>	– display task activity data.....	1259
<b>spyStop()</b>	– stop spying and reporting .....	1260
<b>spyTask()</b>	– run periodic task activity reports .....	1260
<b>sqrt()</b>	– compute a non-negative square root (ANSI).....	1261
<b>sqrtf()</b>	– compute a non-negative square root (ANSI).....	1261
<b>sr()</b>	– return the contents of the status register (68K, SH).....	1262
<b>srand()</b>	– reset the value of the seed used to generate random numbers (ANSI) ...	1262
<b>sscanf()</b>	– read and convert characters from an ASCII string (ANSI).....	1263
<b>stackEntryIsBottom()</b>	– test if an interface has no layers beneath it .....	1267
<b>stackEntryIsTop()</b>	– test if an ifStackTable interface has no layers above .....	1267
<b>stat()</b>	– get file status information using a pathname (POSIX).....	1268
<b>statfs()</b>	– get file status information using a pathname (POSIX).....	1268
<b>stdioFp()</b>	– return the standard input/output/error FILE of the current task.....	1269
<b>stdioInit()</b>	– initialize standard I/O support .....	1269
<b>stdioShow()</b>	– display file pointer internals .....	1270
<b>stdioShowInit()</b>	– initialize the standard I/O show facility .....	1270
<b>strcat()</b>	– concatenate one string to another (ANSI).....	1271
<b>strchr()</b>	– find the first occurrence of a character in a string (ANSI).....	1271
<b>strcmp()</b>	– compare two strings lexicographically (ANSI) .....	1272
<b>strcoll()</b>	– compare two strings as appropriate to <b>LC_COLLATE</b> (ANSI).....	1272
<b>strcpy()</b>	– copy one string to another (ANSI) .....	1273
<b>strcspn()</b>	– return the string length up to first character from a given set (ANSI) ...	1273
<b>strerror()</b>	– map an error number to an error string (ANSI).....	1274
<b>strerror_r()</b>	– map an error number to an error string (POSIX).....	1274
<b>strftime()</b>	– convert broken-down time into a formatted string (ANSI) .....	1275
<b>strlen()</b>	– determine the length of a string (ANSI).....	1277
<b>strncat()</b>	– concatenate characters from one string to another (ANSI) .....	1277
<b>strncmp()</b>	– compare the first n characters of two strings (ANSI).....	1278
<b>strncpy()</b>	– copy characters from one string to another (ANSI).....	1278
<b>strpbrk()</b>	– find first occurrence in a string of a character from a given set (ANSI) ..	1279
<b>strrchr()</b>	– find last occurrence of a character in a string (ANSI) .....	1279
<b>strspn()</b>	– return the string length up to first character not in a given set (ANSI)...	1280
<b>strstr()</b>	– find the first occurrence of a substring in a string (ANSI) .....	1280
<b>strtod()</b>	– convert the initial portion of a string to a double (ANSI).....	1281
<b>strtok()</b>	– break down a string into tokens (ANSI) .....	1282
<b>strtok_r()</b>	– break down a string into tokens (reentrant) (POSIX).....	1283

<b>strtol()</b>	– convert a string to a long integer (ANSI) .....	1284
<b>strtoul()</b>	– convert a string to an unsigned long integer (ANSI) .....	1285
<b>strxfrm()</b>	– transform up to n characters of s2 into s1 (ANSI).....	1287
<b>swab()</b>	– swap bytes.....	1287
<b>symAdd()</b>	– create and add a symbol to a symbol table, including a group number	1288
<b>symByValueAndTypeFind()</b>	– look up a symbol by value and type.....	1289
<b>symByValueFind()</b>	– look up a symbol by value.....	1290
<b>symEach()</b>	– call a routine to examine each entry in a symbol table .....	1290
<b>symFindByName()</b>	– look up a symbol by name.....	1291
<b>symFindByNameAndType()</b>	– look up a symbol by name and type.....	1292
<b>symFindByValue()</b>	– look up a symbol by value.....	1292
<b>symFindByValueAndType()</b>	– look up a symbol by value and type.....	1293
<b>symLibInit()</b>	– initialize the symbol table library.....	1294
<b>symRemove()</b>	– remove a symbol from a symbol table.....	1294
<b>symSyncLibInit()</b>	– initialize host/target symbol table synchronization .....	1295
<b>symSyncTimeoutSet()</b>	– set WTX timeout .....	1295
<b>symTblCreate()</b>	– create a symbol table .....	1296
<b>symTblDelete()</b>	– delete a symbol table .....	1296
<b>sysAuxClkConnect()</b>	– connect a routine to the auxiliary clock interrupt.....	1297
<b>sysAuxClkDisable()</b>	– turn off auxiliary clock interrupts .....	1297
<b>sysAuxClkEnable()</b>	– turn on auxiliary clock interrupts.....	1298
<b>sysAuxClkRateGet()</b>	– get the auxiliary clock rate.....	1298
<b>sysAuxClkRateSet()</b>	– set the auxiliary clock rate .....	1299
<b>sysBspRev()</b>	– return the BSP version and revision number.....	1299
<b>sysBusIntAck()</b>	– acknowledge a bus interrupt .....	1300
<b>sysBusIntGen()</b>	– generate a bus interrupt.....	1300
<b>sysBusTas()</b>	– test and set a location across the bus .....	1301
<b>sysBusToLocalAdrs()</b>	– convert a bus address to a local address .....	1301
<b>sysClkConnect()</b>	– connect a routine to the system clock interrupt .....	1302
<b>sysClkDisable()</b>	– turn off system clock interrupts.....	1302
<b>sysClkEnable()</b>	– turn on system clock interrupts.....	1303
<b>sysClkRateGet()</b>	– get the system clock rate .....	1303
<b>sysClkRateSet()</b>	– set the system clock rate.....	1304
<b>sysHwInit()</b>	– initialize the system hardware.....	1304
<b>sysIntDisable()</b>	– disable a bus interrupt level .....	1305
<b>sysIntEnable()</b>	– enable a bus interrupt level .....	1305
<b>sysLocalToBusAdrs()</b>	– convert a local address to a bus address .....	1306
<b>sysMailboxConnect()</b>	– connect a routine to the mailbox interrupt .....	1306
<b>sysMailboxEnable()</b>	– enable the mailbox interrupt.....	1307
<b>sysMemTop()</b>	– get the address of the top of logical memory .....	1307
<b>sysModel()</b>	– return the model name of the CPU board.....	1308
<b>sysNanoDelay()</b>	– delay for specified number of nanoseconds .....	1308
<b>sysNvRamGet()</b>	– get the contents of non-volatile RAM .....	1309
<b>sysNvRamSet()</b>	– write to non-volatile RAM.....	1310
<b>sysPhysMemTop()</b>	– get the address of the top of memory .....	1310

<code>sysProcNumGet()</code>	– get the processor number .....	1311
<code>sysProcNumSet()</code>	– set the processor number.....	1311
<code>sysScsiBusReset()</code>	– assert the RST line on the SCSI bus (Western Digital WD33C93 only) ...	1312
<code>sysScsiConfig()</code>	– system SCSI configuration .....	1313
<code>sysScsiInit()</code>	– initialize an on-board SCSI port .....	1314
<code>sysSerialChanGet()</code>	– get the <code>SIO_CHAN</code> device associated with a serial channel .....	1315
<code>sysSerialHwInit()</code>	– initialize the BSP serial devices to a quiescent state .....	1315
<code>sysSerialHwInit2()</code>	– connect BSP serial device interrupts.....	1316
<code>sysSerialReset()</code>	– reset all SIO devices to a quiet state.....	1316
<code>system()</code>	– pass a string to a command processor (Unimplemented) (ANSI) .....	1317
<code>sysToMonitor()</code>	– transfer control to the ROM monitor .....	1317
<code>tan()</code>	– compute a tangent (ANSI).....	1318
<code>tanf()</code>	– compute a tangent (ANSI).....	1318
<code>tanh()</code>	– compute a hyperbolic tangent (ANSI).....	1319
<code>tanhf()</code>	– compute a hyperbolic tangent (ANSI).....	1319
<code>tapeFsDevInit()</code>	– associate a sequential device with tape volume functions.....	1320
<code>tapeFsInit()</code>	– initialize the tape volume library .....	1321
<code>tapeFsReadyChange()</code>	– notify <code>tapeFsLib</code> of a change in ready status .....	1321
<code>tapeFsVolUnmount()</code>	– disable a tape device volume .....	1322
<code>tarArchive()</code>	– archive named file/dir onto tape in tar format.....	1322
<code>tarExtract()</code>	– extract all files from a tar formatted tape.....	1323
<code>tarToc()</code>	– display all contents of a tar formatted tape .....	1324
<code>taskActivate()</code>	– activate a task that has been initialized .....	1325
<code>taskCreateHookAdd()</code>	– add a routine to be called at every task create .....	1325
<code>taskCreateHookDelete()</code>	– delete a previously added task create routine.....	1326
<code>taskCreateHookShow()</code>	– show the list of task create routines .....	1326
<code>taskDelay()</code>	– delay a task from executing .....	1327
<code>taskDelete()</code>	– delete a task .....	1327
<code>taskDeleteForce()</code>	– delete a task without restriction .....	1328
<code>taskDeleteHookAdd()</code>	– add a routine to be called at every task delete .....	1329
<code>taskDeleteHookDelete()</code>	– delete a previously added task delete routine .....	1329
<code>taskDeleteHookShow()</code>	– show the list of task delete routines.....	1330
<code>taskHookInit()</code>	– initialize task hook facilities.....	1330
<code>taskHookShowInit()</code>	– initialize the task hook show facility .....	1331
<code>taskIdDefault()</code>	– set the default task ID .....	1331
<code>taskIdListGet()</code>	– get a list of active task IDs .....	1332
<code>taskIdSelf()</code>	– get the task ID of a running task .....	1332
<code>taskIdVerify()</code>	– verify the existence of a task .....	1333
<code>taskInfoGet()</code>	– get information about a task .....	1333
<code>taskInit()</code>	– initialize a task with a stack at a specified address .....	1334
<code>taskIsReady()</code>	– check if a task is ready to run.....	1335
<code>taskIsSuspended()</code>	– check if a task is suspended .....	1335
<code>taskLock()</code>	– disable task rescheduling .....	1336
<code>taskName()</code>	– get the name associated with a task ID .....	1336
<code>taskNameToId()</code>	– look up the task ID associated with a task name.....	1337

<code>taskOptionsGet()</code>	– examine task options .....	1337
<code>taskOptionsSet()</code>	– change task options .....	1338
<code>taskPriorityGet()</code>	– examine the priority of a task.....	1339
<code>taskPrioritySet()</code>	– change the priority of a task.....	1339
<code>taskRegsGet()</code>	– get a task’s registers from the TCB.....	1340
<code>taskRegsSet()</code>	– set a task’s registers .....	1340
<code>taskRegsShow()</code>	– display the contents of a task’s registers .....	1341
<code>taskRestart()</code>	– restart a task.....	1341
<code>taskResume()</code>	– resume a task.....	1342
<code>taskSafe()</code>	– make the calling task safe from deletion .....	1342
<code>taskShow()</code>	– display task information from TCBs.....	1343
<code>taskShowInit()</code>	– initialize the task show routine facility.....	1344
<code>taskSpawn()</code>	– spawn a task.....	1345
<code>taskSRInit()</code>	– initialize the default task status register (MIPS).....	1346
<code>taskSRSet()</code>	– set the task status register (68K, MIPS, x86).....	1347
<code>taskStatusString()</code>	– get a task’s status as a string.....	1347
<code>taskSuspend()</code>	– suspend a task .....	1348
<code>taskSwitchHookAdd()</code>	– add a routine to be called at every task switch.....	1349
<code>taskSwitchHookDelete()</code>	– delete a previously added task switch routine.....	1350
<code>taskSwitchHookShow()</code>	– show the list of task switch routines .....	1350
<code>taskTcb()</code>	– get the task control block for a task ID .....	1351
<code>taskUnlock()</code>	– enable task rescheduling.....	1351
<code>taskUnsafe()</code>	– make the calling task unsafe from deletion.....	1352
<code>taskVarAdd()</code>	– add a task variable to a task .....	1352
<code>taskVarDelete()</code>	– remove a task variable from a task.....	1354
<code>taskVarGet()</code>	– get the value of a task variable.....	1354
<code>taskVarInfo()</code>	– get a list of task variables of a task .....	1355
<code>taskVarInit()</code>	– initialize the task variables facility .....	1355
<code>taskVarSet()</code>	– set the value of a task variable .....	1356
<code>tcpDebugShow()</code>	– display debugging information for the TCP protocol.....	1357
<code>tcpShowInit()</code>	– initialize TCP show routines .....	1357
<code>tcpstatShow()</code>	– display all statistics for the TCP protocol.....	1358
<code>td()</code>	– delete a task.....	1358
<code>telnetdExit()</code>	– close an active <b>telnet</b> session.....	1359
<code>telnetdInit()</code>	– initialize the <b>telnet</b> services .....	1359
<code>telnetdParserSet()</code>	– specify a command interpreter for <b>telnet</b> sessions .....	1360
<code>telnetdStart()</code>	– initialize the <b>telnet</b> services .....	1361
<code>telnetdStaticTaskInitializationGet()</code>	– report whether tasks were pre-started by <b>telnetd</b> .....	1362
<code>tffsBootImagePut()</code>	– write to the boot-image region of the flash device.....	1362
<code>tffsDevCreate()</code>	– create a TrueFFS block device suitable for use with dosFs.....	1363
<code>tffsDevFormat()</code>	– format a flash device for use with TrueFFS .....	1364
<code>tffsDevOptionsSet()</code>	– set TrueFFS volume options .....	1365
<code>tffsDrv()</code>	– initialize the TrueFFS system .....	1365
<code>tffsRawio()</code>	– low level I/O access to flash components.....	1366
<code>tffsShow()</code>	– show device information on a specific socket interface .....	1367

<b>tffsShowAll()</b>	– show device information on all socket interfaces .....	1368
<b>tftpCopy()</b>	– transfer a file via TFTP .....	1368
<b>tftpdDirectoryAdd()</b>	– add a directory to the access list .....	1369
<b>tftpdDirectoryRemove()</b>	– delete a directory from the access list .....	1369
<b>tftpdInit()</b>	– initialize the TFTP server task .....	1370
<b>tftpdTask()</b>	– TFTP server daemon task .....	1371
<b>tftpGet()</b>	– get a file from a remote system .....	1372
<b>tftpInfoShow()</b>	– get TFTP status information .....	1372
<b>tftpInit()</b>	– initialize a TFTP session .....	1373
<b>tftpModeSet()</b>	– set the TFTP transfer mode .....	1373
<b>tftpPeerSet()</b>	– set the TFTP server address .....	1374
<b>tftpPut()</b>	– put a file to a remote system .....	1374
<b>tftpQuit()</b>	– quit a TFTP session .....	1375
<b>tftpSend()</b>	– send a TFTP message to the remote system .....	1375
<b>tftpXfer()</b>	– transfer a file via TFTP using a stream interface .....	1376
<b>ti()</b>	– print complete information from a task's TCB .....	1378
<b>tickAnnounce()</b>	– announce a clock tick to the kernel .....	1379
<b>tickGet()</b>	– get the value of the kernel's tick counter .....	1379
<b>tickSet()</b>	– set the value of the kernel's tick counter .....	1380
<b>time()</b>	– determine the current calendar time (ANSI) .....	1380
<b>timer_cancel()</b>	– cancel a timer .....	1381
<b>timer_connect()</b>	– connect a user routine to the timer signal .....	1381
<b>timer_create()</b>	– allocate a timer using the specified clock for a timing base (POSIX) .....	1382
<b>timer_delete()</b>	– remove a previously created timer (POSIX) .....	1383
<b>timer_getoverrun()</b>	– return the timer expiration overrun (POSIX) .....	1383
<b>timer_gettime()</b>	– get the remaining time before expiration and the reload value (POSIX) .....	1384
<b>timer_settime()</b>	– set the time until the next expiration and arm timer (POSIX) .....	1384
<b>timex()</b>	– time a single execution of a function or functions .....	1385
<b>timexClear()</b>	– clear the list of function calls to be timed .....	1386
<b>timexFunc()</b>	– specify functions to be timed .....	1386
<b>timexHelp()</b>	– display synopsis of execution timer facilities .....	1387
<b>timexInit()</b>	– include the execution timer library .....	1388
<b>timexN()</b>	– time repeated executions of a function or group of functions .....	1388
<b>timexPost()</b>	– specify functions to be called after timing .....	1389
<b>timexPre()</b>	– specify functions to be called prior to timing .....	1390
<b>timexShow()</b>	– display the list of function calls to be timed .....	1390
<b>tmpfile()</b>	– create a temporary binary file (Unimplemented) (ANSI) .....	1391
<b>tmpnam()</b>	– generate a temporary file name (ANSI) .....	1391
<b>tolower()</b>	– convert an upper-case letter to its lower-case equivalent (ANSI) .....	1392
<b>toupper()</b>	– convert a lower-case letter to its upper-case equivalent (ANSI) .....	1392
<b>tr()</b>	– resume a task .....	1393
<b>trgAdd()</b>	– add a new trigger to the trigger list .....	1393
<b>trgChainSet()</b>	– chains two triggers .....	1395
<b>trgDelete()</b>	– delete a trigger from the trigger list .....	1395
<b>trgDisable()</b>	– turn a trigger off .....	1396

<code>trgEnable()</code>	– enable a trigger .....	1396
<code>trgEvent()</code>	– trigger a user-defined event .....	1397
<code>trgLibInit()</code>	– initialize the triggering library .....	1397
<code>trgOff()</code>	– set triggering off .....	1398
<code>trgOn()</code>	– set triggering on .....	1398
<code>trgShow()</code>	– show trigger information .....	1399
<code>trgShowInit()</code>	– initialize the trigger show facility .....	1399
<code>trgWorkQReset()</code>	– reset the trigger work queue task and queue .....	1400
<code>trunc()</code>	– truncate to integer .....	1400
<code>truncf()</code>	– truncate to integer .....	1401
<code>ts()</code>	– suspend a task .....	1401
<code>tsfsUploadPathClose()</code>	– close the TSFS-socket upload path (Windview) .....	1402
<code>tsfsUploadPathCreate()</code>	– open an upload path to the host using a TSFS socket (Windview) .....	1402
<code>tsfsUploadPathLibInit()</code>	– initialize <b>wvTsfsUploadPathLib</b> library (Windview) .....	1403
<code>tsfsUploadPathWrite()</code>	– write to the TSFS upload path (Windview) .....	1403
<code>tt()</code>	– display a stack trace of a task .....	1404
<code>ttyDevCreate()</code>	– create a VxWorks device for a serial channel .....	1405
<code>ttyDrv()</code>	– initialize the tty driver .....	1405
<code>tyAbortFuncSet()</code>	– set the abort function .....	1406
<code>tyAbortSet()</code>	– change the abort character .....	1406
<code>tyBackspaceSet()</code>	– change the backspace character .....	1407
<code>tyDeleteLineSet()</code>	– change the line-delete character .....	1407
<code>tyDevInit()</code>	– initialize the tty device descriptor .....	1408
<code>tyDevRemove()</code>	– remove the tty device descriptor .....	1408
<code>tyEOFSet()</code>	– change the end-of-file character .....	1409
<code>tyIoctl()</code>	– handle device control requests .....	1409
<code>tyIRd()</code>	– interrupt-level input .....	1410
<code>tyITx()</code>	– interrupt-level output .....	1410
<code>tyMonitorTrapSet()</code>	– change the trap-to-monitor character .....	1411
<code>tyRead()</code>	– do a task-level read for a tty device .....	1411
<code>tyWrite()</code>	– do a task-level write for a tty device .....	1412
<code>udpShowInit()</code>	– initialize UDP show routines .....	1413
<code>udpstatShow()</code>	– display statistics for the UDP protocol .....	1413
<code>ungetc()</code>	– push a character back into an input stream (ANSI) .....	1413
<code>unixDiskDevCreate()</code>	– create a UNIX disk device .....	1414
<code>unixDiskInit()</code>	– initialize a dosFs disk on top of UNIX .....	1415
<code>unixDrv()</code>	– install UNIX disk driver .....	1416
<code>unld()</code>	– unload an object module by specifying a file name or module ID .....	1416
<code>unldByGroup()</code>	– unload an object module by specifying a group number .....	1417
<code>unldByModuleId()</code>	– unload an object module by specifying a module ID .....	1417
<code>unldByNameAndPath()</code>	– unload an object module by specifying a name and path .....	1418
<code>unlink()</code>	– delete a file (POSIX) .....	1418
<code>usrAtaConfig()</code>	– mount a DOS file system from an ATA hard disk or a CDROM .....	1419
<code>usrAtaInit()</code>	– initialize the hard disk driver .....	1420
<code>usrClock()</code>	– user-defined system clock interrupt routine .....	1420

<b>usrFdConfig()</b>	– mount a DOS file system from a floppy disk.....	1420
<b>usrFdiskPartCreate()</b>	– create an FDISK-like partition table on a disk .....	1421
<b>usrFdiskPartRead()</b>	– read an FDISK-style partition table .....	1422
<b>usrFdiskPartShow()</b>	– parse and display partition data .....	1423
<b>usrIdeConfig()</b>	– mount a DOS file system from an IDE hard disk.....	1424
<b>usrInit()</b>	– user-defined system initialization routine.....	1424
<b>usrRoot()</b>	– the root task.....	1425
<b>usrScsiConfig()</b>	– configure SCSI peripherals .....	1425
<b>uswab()</b>	– swap bytes with buffers that are not necessarily aligned .....	1426
<b>utime()</b>	– update time on a file .....	1427
<b>va_arg()</b>	– expand to expression having type and value of call’s next argument .....	1428
<b>va_end()</b>	– facilitate a normal return from a routine using a va_list object .....	1428
<b>va_start()</b>	– initialize a va_list object for use by <b>va_arg()</b> and <b>va_end()</b> .....	1429
<b>valloc()</b>	– allocate memory on a page boundary .....	1429
<b>version()</b>	– print VxWorks version information .....	1430
<b>vfdprintf()</b>	– write a string formatted with variable argument list to file descriptor .....	1430
<b>vfprintf()</b>	– write a formatted string to a stream (ANSI) .....	1431
<b>vmBaseGlobalMapInit()</b>	– initialize global mapping .....	1431
<b>vmBaseLibInit()</b>	– initialize base virtual memory support.....	1432
<b>vmBasePageSizeGet()</b>	– return the page size.....	1433
<b>vmBaseStateSet()</b>	– change the state of a block of virtual memory .....	1433
<b>vmContextCreate()</b>	– create a new virtual memory context (VxVMI) .....	1434
<b>vmContextDelete()</b>	– delete a virtual memory context (VxVMI).....	1435
<b>vmContextShow()</b>	– display the translation table for a context (VxVMI).....	1435
<b>vmCurrentGet()</b>	– get the current virtual memory context (VxVMI) .....	1436
<b>vmCurrentSet()</b>	– set the current virtual memory context (VxVMI) .....	1436
<b>vmEnable()</b>	– enable or disable virtual memory (VxVMI) .....	1437
<b>vmGlobalInfoGet()</b>	– get global virtual memory information (VxVMI).....	1437
<b>vmGlobalMap()</b>	– map physical pages to virtual space in shared global virtual mem (VxVMI)..	1438
<b>vmGlobalMapInit()</b>	– initialize global mapping (VxVMI).....	1439
<b>vmLibInit()</b>	– initialize the virtual memory support module (VxVMI).....	1440
<b>vmMap()</b>	– map physical space into virtual space (VxVMI) .....	1440
<b>vmPageBlockSizeGet()</b>	– get the architecture-dependent page block size (VxVMI) .....	1441
<b>vmPageSizeGet()</b>	– return the page size (VxVMI) .....	1442
<b>vmShowInit()</b>	– include virtual memory show facility (VxVMI) .....	1442
<b>vmStateGet()</b>	– get the state of a page of virtual memory (VxVMI) .....	1443
<b>vmStateSet()</b>	– change the state of a block of virtual memory (VxVMI) .....	1444
<b>vmTextProtect()</b>	– write-protect a text segment (VxVMI) .....	1445
<b>vmTranslate()</b>	– translate a virtual address to a physical address (VxVMI) .....	1445
<b>vprintf()</b>	– write string formatted with variable argument list to standard output (ANSI)	1446
<b>vsprintf()</b>	– write a string formatted with a variable argument list to a buffer (ANSI).....	1446
<b>vxCr2Get()</b>	– get a content of the Control Register 2 (x86) .....	1447
<b>vxCr2Set()</b>	– set a value to the Control Register 2 (x86) .....	1447
<b>vxCr3Get()</b>	– get a content of the Control Register 3 (x86) .....	1448
<b>vxCr3Set()</b>	– set a value to the Control Register 3 (x86) .....	1448

<code>vxCr4Get()</code>	– get a content of the Control Register 4 (x86).....	1448
<code>vxCr4Set()</code>	– set a value to the Control Register 4 (x86).....	1449
<code>vxCr0Get()</code>	– get a content of the Control Register 0 (x86).....	1449
<code>vxCr0Set()</code>	– set a value to the Control Register 0 (x86).....	1450
<code>vxDrGet()</code>	– get a content of the Debug Register 0 to 7 (x86).....	1450
<code>vxDrSet()</code>	– set a value to the Debug Register 0 to 7 (x86).....	1451
<code>vxEflagsGet()</code>	– get a content of the EFLAGS register (x86).....	1451
<code>vxEflagsSet()</code>	– set a value to the EFLAGS register (x86).....	1452
<code>vxGdtrGet()</code>	– get a content of the Global Descriptor Table Register (x86).....	1452
<code>vxIdtrGet()</code>	– get a content of the Interrupt Descriptor Table Register (x86).....	1453
<code>vxLdtrGet()</code>	– get a content of the Local Descriptor Table Register (x86).....	1453
<code>vxMemArchProbe()</code>	– architecture-specific part of <code>vxMemProbe()</code> .....	1454
<code>vxMemProbe()</code>	– probe an address for a bus error.....	1454
<code>vxPowerDown()</code>	– place the processor in reduced-power mode (PowerPC, SH).....	1455
<code>vxPowerModeGet()</code>	– get the power management mode (PowerPC, SH, x86).....	1456
<code>vxPowerModeSet()</code>	– set the power management mode (PowerPC, SH, x86).....	1456
<code>vxSSDisable()</code>	– disable the superscalar dispatch (MC68060).....	1458
<code>vxSSEnable()</code>	– enable the superscalar dispatch (MC68060).....	1458
<code>vxTas()</code>	– C-callable atomic test-and-set primitive.....	1459
<code>vxTssGet()</code>	– get a content of the TASK register (x86).....	1460
<code>vxTssSet()</code>	– set a value to the TASK register (x86).....	1460
<code>wcstombs()</code>	– convert a series of wide char’s to multibyte char’s (Unimplemented) (ANSI)	1461
<code>wctomb()</code>	– convert a wide character to a multibyte character (Unimplemented) (ANSI)	1461
<code>wdbSystemSuspend()</code>	– suspend the system.....	1462
<code>wdbUserEvtLibInit()</code>	– include the WDB user event library.....	1463
<code>wdbUserEvtPost()</code>	– post a user event string to host tools.....	1463
<code>wdCancel()</code>	– cancel a currently counting watchdog.....	1464
<code>wdCreate()</code>	– create a watchdog timer.....	1465
<code>wdDelete()</code>	– delete a watchdog timer.....	1465
<code>wdShow()</code>	– show information about a watchdog.....	1466
<code>wdShowInit()</code>	– initialize the watchdog show facility.....	1466
<code>wdStart()</code>	– start a watchdog timer.....	1467
<code>whoami()</code>	– display the current remote identity.....	1468
<code>write()</code>	– write bytes to a file.....	1468
<code>wvEvent()</code>	– log a user-defined event (WindView).....	1469
<code>wvEventInst()</code>	– instrument VxWorks Events (WindView).....	1469
<code>wvEvtBufferGet()</code>	– return the ID of the WindView event buffer (WindView).....	1470
<code>wvEvtClassClear()</code>	– clear a class of events from those being logged (WindView).....	1470
<code>wvEvtClassClearAll()</code>	– clear all classes of events from those logged (WindView).....	1470
<code>wvEvtClassGet()</code>	– get the current set of classes being logged (WindView).....	1471
<code>wvEvtClassSet()</code>	– set the class of events to log (WindView).....	1471
<code>wvEvtLogInit()</code>	– initialize an event log (WindView).....	1472
<code>wvEvtLogStart()</code>	– start logging events to the buffer (WindView).....	1472
<code>wvEvtLogStop()</code>	– stop logging events to the buffer (WindView).....	1473
<code>wvLibInit()</code>	– initialize <code>wvLib</code> - first step (WindView).....	1473

<a href="#">wvLibInit2()</a>	– initialize <b>wvLib</b> - final step (WindView) .....	1473
<a href="#">wvLogHeaderCreate()</a>	– create the event-log header (WindView) .....	1474
<a href="#">wvLogHeaderUpload()</a>	– transfer the log header to the host (WindView).....	1474
<a href="#">wvNetAddressFilterClear()</a>	– remove the address filter for events .....	1475
<a href="#">wvNetAddressFilterSet()</a>	– specify an address filter for events.....	1475
<a href="#">wvNetDisable()</a>	– end reporting of network events to WindView .....	1476
<a href="#">wvNetEnable()</a>	– begin reporting network events to WindView.....	1476
<a href="#">wvNetEventDisable()</a>	– deactivate specific network events .....	1477
<a href="#">wvNetEventEnable()</a>	– activate specific network events.....	1478
<a href="#">wvNetLevelAdd()</a>	– enable network events with specific priority level.....	1479
<a href="#">wvNetLevelRemove()</a>	– disable network events with specific priority level.....	1479
<a href="#">wvNetPortFilterClear()</a>	– remove the port number filter for events.....	1480
<a href="#">wvNetPortFilterSet()</a>	– specify an address filter for events.....	1481
<a href="#">wvObjInst()</a>	– instrument objects (WindView).....	1481
<a href="#">wvObjInstModeSet()</a>	– set object instrumentation on/off (WindView).....	1482
<a href="#">wvRBufMgrPrioritySet()</a>	– set the priority of the WindView rBuf manager (WindView) .....	1483
<a href="#">wvSigInst()</a>	– instrument signals (WindView) .....	1483
<a href="#">wvTaskNamesPreserve()</a>	– preserve an extra copy of task name events (WindView) .....	1484
<a href="#">wvTaskNamesUpload()</a>	– upload preserved task name events (WindView) .....	1485
<a href="#">wvTmrRegister()</a>	– register a timestamp timer (WindView) .....	1485
<a href="#">wvUploadStart()</a>	– start upload of events to the host (WindView) .....	1486
<a href="#">wvUploadStop()</a>	– stop upload of events to host (WindView) .....	1487
<a href="#">wvUploadTaskConfig()</a>	– set priority and stack size of tWVUpload task (WindView).....	1488
<a href="#">xattrib()</a>	– modify MS-DOS file attributes of many files .....	1489
<a href="#">xcopy()</a>	– copy a hierarchy of files with wildcards .....	1490
<a href="#">xdelete()</a>	– delete a hierarchy of files with wildcards .....	1490
<a href="#">zbufCreate()</a>	– create an empty zbuf.....	1492
<a href="#">zbufCut()</a>	– delete bytes from a zbuf.....	1492
<a href="#">zbufDelete()</a>	– delete a zbuf .....	1493
<a href="#">zbufDup()</a>	– duplicate a zbuf.....	1494
<a href="#">zbufExtractCopy()</a>	– copy data from a zbuf to a buffer.....	1495
<a href="#">zbufInsert()</a>	– insert a zbuf into another zbuf.....	1496
<a href="#">zbufInsertBuf()</a>	– create a zbuf segment from a buffer and insert into a zbuf.....	1497
<a href="#">zbufInsertCopy()</a>	– copy buffer data into a zbuf .....	1498
<a href="#">zbufLength()</a>	– determine the length in bytes of a zbuf.....	1499
<a href="#">zbufSegData()</a>	– determine the location of data in a zbuf segment.....	1499
<a href="#">zbufSegFind()</a>	– find the zbuf segment containing a specified byte location .....	1500
<a href="#">zbufSegLength()</a>	– determine the length of a zbuf segment.....	1501
<a href="#">zbufSegNext()</a>	– get the next segment in a zbuf .....	1501
<a href="#">zbufSegPrev()</a>	– get the previous segment in a zbuf .....	1502
<a href="#">zbufSockBufSend()</a>	– create a zbuf from user data and send it to a TCP socket.....	1502
<a href="#">zbufSockBufSendto()</a>	– create a zbuf from a user message and send it to a UDP socket.....	1503
<a href="#">zbufSockLibInit()</a>	– initialize the zbuf socket interface library .....	1505
<a href="#">zbufSockRecv()</a>	– receive data in a zbuf from a TCP socket.....	1505
<a href="#">zbufSockRecvfrom()</a>	– receive a message in a zbuf from a UDP socket.....	1506

<b>zbufSockSend()</b>	- send zbuf data to a TCP socket.....	1507
<b>zbufSockSendto()</b>	- send a zbuf message to a UDP socket.....	1508
<b>zbufSplit()</b>	- split a zbuf into two separate zbufs .....	1509



---

## a0()

<b>NAME</b>	<b>a0()</b> – return the contents of register <b>a0</b> (also <b>a1</b> - <b>a7</b> ) (68K)
<b>SYNOPSIS</b>	<pre>int a0 (     int taskId          /* task ID, 0 means default task */ )</pre>
<b>DESCRIPTION</b>	<p>This command extracts the contents of register <b>a0</b> from the TCB of a specified task. If <i>taskId</i> is omitted or zero, the last task referenced is assumed.</p> <p>Similar routines are provided for all address registers (<b>a0</b> - <b>a7</b>): <b>a0()</b> - <b>a7()</b>.</p> <p>The stack pointer is accessed via <b>a7()</b>.</p>
<b>RETURNS</b>	The contents of register <b>a0</b> (or the requested register).
<b>SEE ALSO</b>	<b>dbgArchLib</b> , <i>VxWorks Programmer's Guide: Target Shell</i>

---

## abort()

<b>NAME</b>	<b>abort()</b> – cause abnormal program termination (ANSI)
<b>SYNOPSIS</b>	<pre>void abort (void)</pre>
<b>DESCRIPTION</b>	<p>This routine causes abnormal program termination, unless the signal <b>SIGABRT</b> is being caught and the signal handler does not return. VxWorks does not flush output streams, close open streams, or remove temporary files. <b>abort()</b> returns unsuccessful status termination to the host environment by calling:</p> <pre>raise (SIGABRT);</pre>
<b>INCLUDE FILES</b>	<b>stdlib.h</b>
<b>RETURNS</b>	This routine cannot return to the caller.
<b>SEE ALSO</b>	<b>ansiStdlib</b>

**abs()**

---

**abs()**

<b>NAME</b>	<b>abs()</b> – compute the absolute value of an integer (ANSI)
<b>SYNOPSIS</b>	<pre>int abs (     int i                /* integer for which to return absolute value */ )</pre>
<b>DESCRIPTION</b>	This routine computes the absolute value of a specified integer. If the result cannot be represented, the behavior is undefined.
<b>INCLUDE FILES</b>	stdlib.h
<b>RETURNS</b>	The absolute value of <i>i</i> .
<b>SEE ALSO</b>	ansiStdlib

---

**accept()**

<b>NAME</b>	<b>accept()</b> – accept a connection from a socket
<b>SYNOPSIS</b>	<pre>int accept (     int s,                /* socket descriptor */     struct sockaddr * addr, /* peer address */     int * addrLen /* peer address length */ )</pre>
<b>DESCRIPTION</b>	<p>This routine accepts a connection on a socket, and returns a new socket created for the connection. The socket must be bound to an address with <b>bind()</b>, and enabled for connections by a call to <b>listen()</b>. The <b>accept()</b> routine dequeues the first connection and creates a new socket with the same properties as <i>s</i>. It blocks the caller until a connection is present, unless the socket is marked as non-blocking.</p> <p>The <i>addrLen</i> parameter should be initialized to the size of the available buffer pointed to by <i>addr</i>. Upon return, <i>addrLen</i> contains the size in bytes of the peer's address stored in <i>addr</i>.</p> <p><b>WARNING:</b> You must make sure that you do not close the file descriptor on which a task is pending during an <b>accept()</b>. Although the <b>accept()</b> on the closed file descriptor</p>

sometimes returns with an error, the **accept()** can also fail to return at all. Thus, if you need to be able to close a socket connections file descriptor asynchronously, you may need to set up a semaphore-based locking mechanism that prevents the close while an **accept()** is pending on the file descriptor.

---

**RETURNS** A socket descriptor, or **ERROR** if the call fails.

**SEE ALSO** **sockLib**

---

## **acos()**

**NAME** **acos()** – compute an arc cosine (ANSI)

**SYNOPSIS**

```
double acos
(
    double x                /* number between -1 and 1 */
)
```

**DESCRIPTION** This routine returns principal value of the arc cosine of  $x$  in double precision (IEEE double, 53 bits). If  $x$  is the cosine of an angle  $T$ , this function returns  $T$ .

A domain error occurs for arguments not in the range  $[-1,+1]$ .

**INCLUDE FILES** **math.h**

**RETURNS** The double-precision arc cosine of  $x$  in the range  $[0,\pi]$  radians.

Special cases:

If  $x$  is NaN, **acos()** returns  $x$ .

If  $|x| > 1$ , it returns NaN.

**SEE ALSO** **ansiMath, mathALib**

## acosf()

<b>NAME</b>	<b>acosf()</b> – compute an arc cosine (ANSI)
<b>SYNOPSIS</b>	<pre>float acosf (     float x    /* number between -1 and 1 */ )</pre>
<b>DESCRIPTION</b>	This routine computes the arc cosine of <i>x</i> in single precision. If <i>x</i> is the cosine of an angle <i>T</i> , this function returns <i>T</i> .
<b>INCLUDE FILES</b>	<b>math.h</b>
<b>RETURNS</b>	The single-precision arc cosine of <i>x</i> in the range 0 to pi radians.
<b>SEE ALSO</b>	<b>mathALib</b>

---

## aioPxLibInit()

<b>NAME</b>	<b>aioPxLibInit()</b> – initialize the asynchronous I/O (AIO) library
<b>SYNOPSIS</b>	<pre>STATUS aioPxLibInit (     int lioMax          /* max outstanding lio calls */ )</pre>
<b>DESCRIPTION</b>	This routine initializes the AIO library. It should be called only once after the I/O system has been initialized. <i>lioMax</i> specifies the maximum number of outstanding <b>lio_listio()</b> calls at one time. If <i>lioMax</i> is zero, the default value of <b>AIO_CLUST_MAX</b> is used.
<b>RETURNS</b>	OK if successful, otherwise <b>ERROR</b> .
<b>ERRNO</b>	<b>S_aioPxLib_IOS_NOT_INITIALIZED</b>
<b>SEE ALSO</b>	<b>aioPxLib</b>

---

## aioShow()

<b>NAME</b>	<b>aioShow()</b> – show AIO requests
<b>SYNOPSIS</b>	<pre><b>STATUS</b> aioShow (     int drvNum          /* drv num to show (IGNORED) */ )</pre>
<b>DESCRIPTION</b>	This routine displays the outstanding AIO requests. <hr/> <b>WARNING:</b> The <i>drvNum</i> parameter is not currently used. <hr/>
<b>RETURNS</b>	OK, always.
<b>SEE ALSO</b>	<b>aioPxShow</b>

---

## aioSysInit()

<b>NAME</b>	<b>aioSysInit()</b> – initialize the AIO system driver
<b>SYNOPSIS</b>	<pre><b>STATUS</b> aioSysInit (     int numTasks,      /* number of system tasks */     int taskPrio,      /* AIO task priority */     int taskStackSize /* AIO task stack size */ )</pre>
<b>DESCRIPTION</b>	This routine initializes the AIO system driver. It should be called once after the AIO library has been initialized. It spawns <i>numTasks</i> system I/O tasks to be executed at <i>taskPrio</i> priority level, with a stack size of <i>taskStackSize</i> . It also starts the wait task and sets the system driver as the default driver for AIO. If <i>numTasks</i> , <i>taskPrio</i> , or <i>taskStackSize</i> is 0, a default value (AIO_IO_TASKS_DFLT, AIO_IO_PRIO_DFLT, or AIO_IO_STACK_DFLT, respectively) is used.
<b>RETURNS</b>	OK if successful, otherwise <b>ERROR</b> .
<b>SEE ALSO</b>	<b>aioSysDrv</b>

## **aio\_error()**

<b>NAME</b>	<b>aio_error()</b> – retrieve error status of asynchronous I/O operation (POSIX)
<b>SYNOPSIS</b>	<pre>int aio_error (     const struct aiocb * pAiocb /* AIO control block */ )</pre>
<b>DESCRIPTION</b>	This routine returns the error status associated with the I/O operation specified by <i>pAiocb</i> . If the operation is not yet completed, the error status will be <b>EINPROGRESS</b> .
<b>RETURNS</b>	<b>EINPROGRESS</b> if the AIO operation has not yet completed, <b>OK</b> if the AIO operation completed successfully, the error status if the AIO operation failed, otherwise <b>ERROR</b> .
<b>ERRNO</b>	<b>EINVAL</b>
<b>INCLUDE FILES</b>	<b>aio.h</b>
<b>SEE ALSO</b>	<b>aioPxLib</b>

---

## **aio\_read()**

<b>NAME</b>	<b>aio_read()</b> – initiate an asynchronous read (POSIX)
<b>SYNOPSIS</b>	<pre>int aio_read (     struct aiocb * pAiocb /* AIO control block */ )</pre>
<b>DESCRIPTION</b>	<p>This routine asynchronously reads data based on the following parameters specified by members of the AIO control structure <i>pAiocb</i>. It reads <b>aio_nbytes</b> bytes of data from the file <b>aio_fildes</b> into the buffer <b>aio_buf</b>.</p> <p>The requested operation takes place at the absolute position in the file as specified by <b>aio_offset</b>.</p> <p><b>aio_reqprio</b> can be used to lower the priority of the AIO request; if this parameter is nonzero, the priority of the AIO request is <b>aio_reqprio</b> lower than the calling task priority.</p>

The call returns when the read request has been initiated or queued to the device. **aio\_error()** can be used to determine the error status and of the AIO operation. On completion, **aio\_return()** can be used to determine the return status.

**aio\_sigevent** defines the signal to be generated on completion of the read request. If this value is zero, no signal is generated.

<b>RETURNS</b>	OK if the read queued successfully, otherwise <b>ERROR</b> .
<b>ERRNO</b>	EBADF, EINVAL
<b>INCLUDE FILES</b>	<b>aio.h</b>
<b>SEE ALSO</b>	<b>aioPxLib</b> , <b>aio_error()</b> , <b>aio_return()</b> , <b>read()</b>

---

## **aio\_return()**

<b>NAME</b>	<b>aio_return()</b> – retrieve return status of asynchronous I/O operation (POSIX)
<b>SYNOPSIS</b>	<pre>size_t aio_return (     struct aiocb * pAiocb    /* AIO control block */ )</pre>
<b>DESCRIPTION</b>	This routine returns the return status associated with the I/O operation specified by <i>pAiocb</i> . The return status for an AIO operation is the value that would be returned by the corresponding <b>read()</b> , <b>write()</b> , or <b>fsync()</b> call. <b>aio_return()</b> may be called only after the AIO operation has completed ( <b>aio_error()</b> returns a valid error code--not <b>EINPROGRESS</b> ). Furthermore, <b>aio_return()</b> may be called only once; subsequent calls will fail.
<b>RETURNS</b>	The return status of the completed AIO request, or <b>ERROR</b> .
<b>ERRNO</b>	<b>EINVAL</b> , <b>EINPROGRESS</b>
<b>INCLUDE FILES</b>	<b>aio.h</b>
<b>SEE ALSO</b>	<b>aioPxLib</b>

## **aio\_suspend()**

- NAME** `aio_suspend()` – wait for asynchronous I/O request(s) (POSIX)
- SYNOPSIS**
- ```
int aio_suspend
(
    const struct aiocb * list[], /* AIO requests */
    int nEnt, /* number of requests */
    const struct timespec * timeout /* wait timeout */
)
```
- DESCRIPTION** This routine suspends the caller until one of the following occurs:
- at least one of the previously submitted asynchronous I/O operations referenced by *list* has completed,
  - a signal interrupts the function, or
  - the time interval specified by *timeout* has passed (if *timeout* is not NULL).
- RETURNS** OK if an AIO request completes, otherwise ERROR.
- ERRNO** EAGAIN, EINTR
- INCLUDE FILES** `aio.h`
- SEE ALSO** `aioPxBLib`
- 

## **aio\_write()**

- NAME** `aio_write()` – initiate an asynchronous write (POSIX)
- SYNOPSIS**
- ```
int aio_write
(
    struct aiocb * pAiocb /* AIO control block */
)
```
- DESCRIPTION** This routine asynchronously writes data based on the following parameters specified by members of the AIO control structure *pAiocb*. It writes **aio\_nbytes** of data to the file **aio\_fildes** from the buffer **aio\_buf**.

The requested operation takes place at the absolute position in the file as specified by **aio\_offset**.

**aio\_reqprio** can be used to lower the priority of the AIO request; if this parameter is nonzero, the priority of the AIO request is **aio\_reqprio** lower than the calling task priority.

The call returns when the write request has been initiated or queued to the device. **aio\_error()** can be used to determine the error status and of the AIO operation. On completion, **aio\_return()** can be used to determine the return status.

**aio\_sigevent** defines the signal to be generated on completion of the write request. If this value is zero, no signal is generated.

<b>RETURNS</b>	OK if write queued successfully, otherwise <b>ERROR</b> .
<b>ERRNO</b>	<b>EBADF</b> , <b>EINVAL</b>
<b>INCLUDE FILES</b>	<b>aio.h</b>
<b>SEE ALSO</b>	<b>aioPxLib</b> , <b>aio_error()</b> , <b>aio_return()</b> , <b>write()</b>

---

## alarm()

<b>NAME</b>	<b>alarm()</b> – set an alarm clock for delivery of a signal
<b>SYNOPSIS</b>	<pre><b>unsigned int alarm</b> (<b>unsigned int secs</b>)</pre>
<b>DESCRIPTION</b>	<p>This routine arranges for a <b>SIGALRM</b> signal to be delivered to the calling task after <i>secs</i> seconds.</p> <p>If <i>secs</i> is zero, no new alarm is scheduled. In all cases, any previously set alarm is cancelled.</p>
<b>RETURNS</b>	Time remaining until a previously scheduled alarm was due to be delivered, zero if there was no previous alarm, or <b>ERROR</b> in case of an error.
<b>SEE ALSO</b>	<b>timerLib</b>

---

## arpAdd()

**NAME**            **arpAdd()** – create or modify an ARP table entry

**SYNOPSIS**        **STATUS** arpAdd  
                  (  
                  char \* pHost,                 /\* host name or IP address \*/  
                  char \* pEther,               /\* Ethernet address \*/  
                  int    flags                 /\* ARP flags \*/  
                  )

**DESCRIPTION**    This routine assigns an Ethernet address to an IP address in the ARP table. The *pHost* parameter specifies the host by name or by Internet address using standard dotted decimal notation. The *pEther* parameter provides the Ethernet address as six hexadecimal bytes (between 0 and ff) separated by colons. A new entry is created for the specified host if necessary. Otherwise, the existing entry is changed to use the given Ethernet address.

The *flags* parameter combines any of the following options:

**ATF\_PERM** (0x04)

Create a permanent ARP entry which will not time out.

**ATF\_PUBL** (0x08)

Publish this entry. The host will respond to ARP requests even if the *pHost* parameter does not match a local IP address. This setting provides a limited form of proxy ARP.

**ATF\_PROXY** (0x10)

Use a "wildcard" hardware address. The proxy server uses this setting to support multiple proxy networks. The entry always supplies the hardware address of the sending interface.

**EXAMPLE**        Create a permanent ARP table entry for "myHost" with Ethernet address 0:80:f9:1:2:3:

```
arpAdd ("myHost", "0:80:f9:1:2:3", 0x4);
```

Assuming "myHost" has the Internet address "90.0.0.3", the following call changes the Ethernet address to 0:80:f9:1:2:4. No additional flags are set for that entry.

```
arpAdd ("90.0.0.3", "0:80:f9:1:2:4", 0);
```

**RETURNS**        OK, or ERROR if unsuccessful.

**ERRNO**            S\_arpLib\_INVALID\_ARGUMENT  
                  S\_arpLib\_INVALID\_HOST  
                  S\_arpLib\_INVALID\_ENET\_ADDRESS  
                  S\_arpLib\_INVALID\_FLAG  
                  or results of low-level ioctl call.

SEE ALSO `arpLib`

---

## `arpDelete()`

**NAME** `arpDelete()` – remove an ARP table entry

**SYNOPSIS**

```
STATUS arpDelete
(
    char * pHost          /* host name or IP address */
)
```

**DESCRIPTION** This routine deletes an ARP table entry. The *pHost* parameter indicates the target entry using the host name or Internet address.

**EXAMPLE**

```
arpDelete ("91.0.0.3")
arpDelete ("myHost")
```

**RETURNS** OK, or ERROR if unsuccessful.

**ERRNO** `S_arpLib_INVALID_ARGUMENT`  
`S_arpLib_INVALID_HOST`

SEE ALSO `arpLib`

---

## `arpFlush()`

**NAME** `arpFlush()` – flush all entries in the system ARP table

**SYNOPSIS**

```
void arpFlush (void)
```

**DESCRIPTION** This routine flushes all non-permanent entries in the ARP cache.

**RETURNS** N/A

SEE ALSO `arpLib`

## arpResolve()

**NAME** arpResolve() – resolve a hardware address for a specified Internet address

**SYNOPSIS**

```
STATUS arpResolve
(
    char * targetAddr,      /* name or Internet address of target */
    char * pHwAddr,        /* where to return the H/W address */
    int   numTries,        /* number of times to try ARPing (-1 means */
                          /* try forever) */
    int   numTicks         /* number of ticks between ARPs */
)
```

**DESCRIPTION** This routine uses the Address Resolution Protocol (ARP) and internal ARP cache to resolve the hardware address of a machine that owns the Internet address given in *targetAddr*.

The hardware address is copied to *pHwAddr* as network byte order, if the resolution of *targetAddr* is successful. *pHwAddr* must point to a buffer which is large enough to receive the address.

---

**NOTE:** RFC 1122 prohibits sending more than one arp request per second. Any numTicks value that would result in a shorter time than this is ignored.

---

**RETURNS** OK if the address is resolved successfully, or ERROR if *pHwAddr* is NULL, *targetAddr* is invalid, or address resolution is unsuccessful.

**ERRNO** S\_arpLib\_INVALID\_ARGUMENT  
S\_arpLib\_INVALID\_HOST

**SEE ALSO** arpLib

---

## arpShow()

**NAME** `arpShow()` – display entries in the system ARP table

**SYNOPSIS** `void arpShow (void)`

**DESCRIPTION** This routine displays the current Internet-to-Ethernet address mappings in the ARP table.

**EXAMPLE**

```
-> arpShow
LINK LEVEL ARP TABLE
Destination      LL Address      Flags  Refcnt Use      Interface
-----
90.0.0.63        08:00:3e:23:79:e7 0x405  0      82      lo0
-----
```

Some configuration is required when this routine is to be used remotely over the network, *e.g.*, through a telnet session or through the host shell using `WDB_COMM_NETWORK`. If more than 5 entries are expected in the table the parameter `RT_BUFFERED_DISPLAY` should be set to `TRUE` to prevent a possible deadlock. This requires a buffer whose size can be set with `RT_DISPLAY_MEMORY`. It will limit the number of entries that can be displayed (each entry requires approx. 70 bytes).

**RETURNS** N/A

**SEE ALSO** `netShow`

---

## arptabShow()

**NAME** `arptabShow()` – display the known ARP entries

**SYNOPSIS** `void arptabShow (void)`

**DESCRIPTION** This routine displays current Internet-to-Ethernet address mappings in the ARP table.

**RETURNS** N/A

**SEE ALSO** `netShow`

## asctime()

**NAME** `asctime()` – convert broken-down time into a string (ANSI)

**SYNOPSIS**

```
char * asctime
(
    const struct tm * timeptr /* broken-down time */
)
```

**DESCRIPTION** This routine converts the broken-down time pointed to by *timeptr* into a string of the form:

```
SUN SEP 16 01:03:52 1973\n\0
```

This routine is not reentrant. For a reentrant version, see `asctime_r()`.

**INCLUDE FILES** `time.h`

**RETURNS** A pointer to the created string.

**SEE ALSO** `ansiTime`

---

## asctime\_r()

**NAME** `asctime_r()` – convert broken-down time into a string (POSIX)

**SYNOPSIS**

```
int asctime_r
(
    const struct tm * timeptr, /* broken-down time */
    char *          asctimeBuf, /* buffer to contain string */
    size_t *        buflen     /* size of buffer */
)
```

**DESCRIPTION** This routine converts the broken-down time pointed to by *timeptr* into a string of the form:

```
SUN SEP 16 01:03:52 1973\n\0
```

The string is copied to *asctimeBuf*. `asctimer()` is the POSIX re-entrant version of `asctime()`.

**INCLUDE FILES** `time.h`

**RETURNS** The size of the created string.

**SEE ALSO** `ansiTime`

---

## asin()

<b>NAME</b>	<b>asin()</b> – compute an arc sine (ANSI)
<b>SYNOPSIS</b>	<pre>double asin (     double x          /* number between -1 and 1 */ )</pre>
<b>DESCRIPTION</b>	<p>This routine returns the principal value of the arc sine of <math>x</math> in double precision (IEEE double, 53 bits). If <math>x</math> is the sine of an angle <math>T</math>, this function returns <math>T</math>.</p> <p>A domain error occurs for arguments not in the range <math>[-1,+1]</math>.</p>
<b>INCLUDE FILES</b>	<b>math.h</b>
<b>RETURNS</b>	<p>The double-precision arc sine of <math>x</math> in the range <math>[-\pi/2,\pi/2]</math> radians.</p> <p>Special cases: If <math>x</math> is NaN, <b>asin()</b> returns <math>x</math>. If <math> x &gt;1</math>, it returns NaN.</p>
<b>SEE ALSO</b>	<b>ansiMath, mathALib</b>

---

## asinf()

<b>NAME</b>	<b>asinf()</b> – compute an arc sine (ANSI)
<b>SYNOPSIS</b>	<pre>float asinf (     float x   /* number between -1 and 1 */ )</pre>
<b>DESCRIPTION</b>	<p>This routine computes the arc sine of <math>x</math> in single precision. If <math>x</math> is the sine of an angle <math>T</math>, this function returns <math>T</math>.</p>
<b>INCLUDE FILES</b>	<b>math.h</b>
<b>RETURNS</b>	The single-precision arc sine of $x$ in the range $-\pi/2$ to $\pi/2$ radians.
<b>SEE ALSO</b>	<b>mathALib</b>

## **assert()**

<b>NAME</b>	<b>assert()</b> – put diagnostics into programs (ANSI)
<b>SYNOPSIS</b>	<pre>void assert (     int a )</pre>
<b>DESCRIPTION</b>	<p>If an expression is false (that is, equal to zero), the <b>assert()</b> macro writes information about the failed call to standard error in an implementation-defined format. It then calls <b>abort()</b>. The diagnostic information includes:</p> <ul style="list-style-type: none"><li>- the text of the argument</li><li>- the name of the source file (value of preprocessor macro <code>__FILE__</code>)</li><li>- the source line number (value of preprocessor macro <code>__LINE__</code>)</li></ul>
<b>INCLUDE</b>	<b>stdio.h, stdlib.h, assert.h</b>
<b>RETURNS</b>	N/A
<b>SEE ALSO</b>	<b>ansiAssert</b>

---

## **atan()**

<b>NAME</b>	<b>atan()</b> – compute an arc tangent (ANSI)
<b>SYNOPSIS</b>	<pre>double atan (     double x                /* tangent of an angle */ )</pre>
<b>DESCRIPTION</b>	<p>This routine returns the principal value of the arc tangent of <math>x</math> in double precision (IEEE double, 53 bits). If <math>x</math> is the tangent of an angle <math>T</math>, this function returns <math>T</math> (in radians).</p>
<b>INCLUDE FILES</b>	<b>math.h</b>
<b>RETURNS</b>	The double-precision arc tangent of $x$ in the range $[-\pi/2, \pi/2]$ radians. Special case: if $x$ is NaN, <b>atan()</b> returns $x$ itself.
<b>SEE ALSO</b>	<b>ansiMath, mathALib</b>

---

## atan2()

**NAME** `atan2()` – compute the arc tangent of  $y/x$  (ANSI)

**SYNOPSIS**

```
double atan2
(
    double y,          /* numerator */
    double x          /* denominator */
)
```

**DESCRIPTION** This routine returns the principal value of the arc tangent of  $y/x$  in double precision (IEEE double, 53 bits). This routine uses the signs of both arguments to determine the quadrant of the return value. A domain error may occur if both arguments are zero.

**INCLUDE FILES** `math.h`

**RETURNS** The double-precision arc tangent of  $y/x$ , in the range  $[-\pi, \pi]$  radians.

Special cases:

Notations:  $\text{atan2}(y,x) == \text{ARG}(x+iy) == \text{ARG}(x,y)$ .

$\text{ARG}(\text{NaN}, (\text{anything}))$	is	NaN
$\text{ARG}((\text{anything}), \text{NaN})$	is	NaN
$\text{ARG}+(\text{anything but NaN}), +-0)$	is	$+-0$
$\text{ARG}-(\text{anything but NaN}), +-0)$	is	$+-\pi$
$\text{ARG}(0, +-(\text{anything but } 0 \text{ and NaN}))$	is	$+-\pi/2$
$\text{ARG}(+\text{INF}, +-(\text{anything but INF and NaN}))$	is	$+-0$
$\text{ARG}(-\text{INF}, +-(\text{anything but INF and NaN}))$	is	$+-\pi$
$\text{ARG}(+\text{INF}, +-\text{INF})$	is	$+-\pi/4$
$\text{ARG}(-\text{INF}, +-\text{INF})$	is	$+-3\pi/4$
$\text{ARG}((\text{anything but } 0, \text{NaN, and INF}), +-\text{INF})$	is	$+-\pi/2$

**SEE ALSO** `ansiMath`, `mathALib`

## **atan2f()**

<b>NAME</b>	<b>atan2f()</b> – compute the arc tangent of $y/x$ (ANSI)
<b>SYNOPSIS</b>	<pre>float atan2f (     float y,          /* numerator */     float x          /* denominator */ )</pre>
<b>DESCRIPTION</b>	This routine returns the principal value of the arc tangent of $y/x$ in single precision.
<b>INCLUDE FILES</b>	<b>math.h</b>
<b>RETURNS</b>	The single-precision arc tangent of $y/x$ in the range $-\pi$ to $\pi$ .
<b>SEE ALSO</b>	<b>mathALib</b>

---

## **atanf()**

<b>NAME</b>	<b>atanf()</b> – compute an arc tangent (ANSI)
<b>SYNOPSIS</b>	<pre>float atanf (     float x  /* tangent of an angle */ )</pre>
<b>DESCRIPTION</b>	This routine computes the arc tangent of $x$ in single precision. If $x$ is the tangent of an angle $T$ , this function returns $T$ (in radians).
<b>INCLUDE FILES</b>	<b>math.h</b>
<b>RETURNS</b>	The single-precision arc tangent of $x$ in the range $-\pi/2$ to $\pi/2$ .
<b>SEE ALSO</b>	<b>mathALib</b>

---

## atexit()

<b>NAME</b>	<b>atexit()</b> – call a function at program termination (Unimplemented) (ANSI)
<b>SYNOPSIS</b>	<pre>int atexit (     void (* __func)(void)    /* pointer to a function */ )</pre>
<b>DESCRIPTION</b>	This routine is unimplemented. VxWorks task exit hooks provide this functionality.
<b>INCLUDE FILES</b>	stdlib.h
<b>RETURNS</b>	ERROR, always.
<b>SEE ALSO</b>	ansiStdlib, taskHookLib

---

## atof()

<b>NAME</b>	<b>atof()</b> – convert a string to a <b>double</b> (ANSI)
<b>SYNOPSIS</b>	<pre>double atof (     const char * s           /* pointer to string */ )</pre>
<b>DESCRIPTION</b>	This routine converts the initial portion of the string <i>s</i> to double-precision representation. Its behavior is equivalent to: <pre>strtod (s, (char **) NULL);</pre>
<b>INCLUDE FILES</b>	stdlib.h
<b>RETURNS</b>	The converted value in double-precision representation.
<b>SEE ALSO</b>	ansiStdlib

**atoi()**

---

**atoi()**

<b>NAME</b>	<b>atoi()</b> – convert a string to an <b>int</b> (ANSI)
<b>SYNOPSIS</b>	<pre>int atoi (     const char * s          /* pointer to string */ )</pre>
<b>DESCRIPTION</b>	This routine converts the initial portion of the string <i>s</i> to <b>int</b> representation. Its behavior is equivalent to: <pre>(int) strtol (s, (char **) NULL, 10);</pre>
<b>INCLUDE FILES</b>	<b>stdlib.h</b>
<b>RETURNS</b>	The converted value represented as an <b>int</b> .
<b>SEE ALSO</b>	<b>ansiStdlib</b>

---

**atol()**

<b>NAME</b>	<b>atol()</b> – convert a string to a <b>long</b> (ANSI)
<b>SYNOPSIS</b>	<pre>long atol (     const register char * s /* pointer to string */ )</pre>
<b>DESCRIPTION</b>	This routine converts the initial portion of the string <i>s</i> to long integer representation. Its behavior is equivalent to: <pre>strtol (s, (char **) NULL, 10);</pre>
<b>INCLUDE FILES</b>	<b>stdlib.h</b>
<b>RETURNS</b>	The converted value represented as a <b>long</b> .
<b>SEE ALSO</b>	<b>ansiStdlib</b>

---

## attrib()

- NAME** `attrib()` – modify MS-DOS file attributes on a file or directory
- SYNOPSIS**
- ```
STATUS attrib
(
    const char * fileName,    /* file or dir name on which to change flags */
    const char * attr       /* flag settings to change */
)
```
- DESCRIPTION**
- This function provides means for the user to modify the attributes of a single file or directory. There are four attribute flags which may be modified: "Archive", "System", "Hidden" and "Read-only". Among these flags, only "Read-only" has a meaning in VxWorks, namely, read-only files can not be modified deleted or renamed.
- The *attr* argument string may contain must start with either "+" or "-", meaning the attribute flags which will follow should be either set or cleared. After "+" or "-" any of these four letter will signify their respective attribute flags - "A", "S", "H" and "R".
- For example, to write-protect a particular file and flag that it is a system file:
- ```
-> attrib( "bootrom.sys", "+RS")
```
- RETURNS** OK, or ERROR if the file can not be opened.
- SEE ALSO** `usrFsLib`

**b()****b()**

**NAME**            **b()** – set or display breakpoints

**SYNOPSIS**        **STATUS** **b**

```
(
    INSTR * addr,           /* where to set breakpoint, or 0 = display */
                          /* all breakpoints */
    int    task,           /* task for which to set breakpoint, 0 = */
                          /* set all tasks */
    int    count,          /* number of passes before hit */
    BOOL   quiet           /* TRUE = don't print debugging info, FALSE */
                          /* = print debugging info */
)
```

**DESCRIPTION**    This routine sets or displays breakpoints. To display the list of currently active breakpoints, call **b()** without arguments:

```
-> b
```

The list shows the address, task, and pass count of each breakpoint. Temporary breakpoints inserted by **so()** and **cret()** are also indicated.

To set a breakpoint with **b()**, include the address, which can be specified numerically or symbolically with an optional offset. The other arguments are optional:

```
-> b addr [, task [, count [, quiet]]]
```

If *task* is zero or omitted, the breakpoint will apply to all breakable tasks. If *count* is zero or omitted, the breakpoint will occur every time it is hit. If *count* is specified, the break will not occur until the *count* +1th time an eligible task hits the breakpoint (*i.e.*, the breakpoint is ignored the first *count* times it is hit).

If *quiet* is specified, debugging information destined for the console will be suppressed when the breakpoint is hit. This option is included for use by external source code debuggers that handle the breakpoint user interface themselves.

Individual tasks can be unbreakable, in which case breakpoints that otherwise would apply to a task are ignored. Tasks can be spawned unbreakable by specifying the task option **VX\_UNBREAKABLE**. Tasks can also be set unbreakable or breakable by resetting **VX\_UNBREAKABLE** with the routine **taskOptionsSet()**.

**RETURNS**        **OK**, or **ERROR** if *addr* is illegal or the breakpoint table is full.

**SEE ALSO**        **dbgLib**, **bd()**, **taskOptionsSet()**, *VxWorks Programmer's Guide: Target Shell*, **windsh**, *Tornado User's Guide: Shell*

---

## **bcmp()**

<b>NAME</b>	<b>bcmp()</b> – compare one buffer to another
<b>SYNOPSIS</b>	<pre>int bcmp (     char * buf1,          /* pointer to first buffer */     char * buf2,          /* pointer to second buffer */     int   nbytes          /* number of bytes to compare */ )</pre>
<b>DESCRIPTION</b>	This routine compares the first <i>nbytes</i> characters of <i>buf1</i> to <i>buf2</i> .
<b>RETURNS</b>	0 if the first <i>nbytes</i> of <i>buf1</i> and <i>buf2</i> are identical, less than 0 if <i>buf1</i> is less than <i>buf2</i> , or greater than 0 if <i>buf1</i> is greater than <i>buf2</i> .
<b>SEE ALSO</b>	<b>bLib</b>

---

## **bcopy()**

<b>NAME</b>	<b>bcopy()</b> – copy one buffer to another
<b>SYNOPSIS</b>	<pre>void bcopy (     const char * source,  /* pointer to source buffer */     char *      destination, /* pointer to destination buffer */     int        nbytes     /* number of bytes to copy */ )</pre>
<b>DESCRIPTION</b>	This routine copies the first <i>nbytes</i> characters from <i>source</i> to <i>destination</i> . Overlapping buffers are handled correctly. Copying is done in the most efficient way possible, which may include long-word, or even multiple-long-word moves on some architectures. In general, the copy will be significantly faster if both buffers are long-word aligned. (For copying that is restricted to byte, word, or long-word moves, see the manual entries for <b>bcopyBytes()</b> , <b>bcopyWords()</b> , and <b>bcopyLongs()</b> .)
<b>RETURNS</b>	N/A
<b>SEE ALSO</b>	<b>bLib</b> , <b>bcopyBytes()</b> , <b>bcopyWords()</b> , <b>bcopyLongs()</b>

## **bcopyBytes()**

**NAME** `bcopyBytes()` – copy one buffer to another one byte at a time

**SYNOPSIS**

```
void bcopyBytes
(
    char * source,          /* pointer to source buffer */
    char * destination,    /* pointer to destination buffer */
    int  nbytes            /* number of bytes to copy */
)
```

**DESCRIPTION** This routine copies the first *nbytes* characters from *source* to *destination* one byte at a time. This may be desirable if a buffer can only be accessed with byte instructions, as in certain byte-wide memory-mapped peripherals.

**RETURNS** N/A

**SEE ALSO** `bLib`, `bcopy()`

---

## **bcopyLongs()**

**NAME** `bcopyLongs()` – copy one buffer to another one long word at a time

**SYNOPSIS**

```
void bcopyLongs
(
    char * source,          /* pointer to source buffer */
    char * destination,    /* pointer to destination buffer */
    int  nlongs            /* number of longs to copy */
)
```

**DESCRIPTION** This routine copies the first *nlongs* characters from *source* to *destination* one long word at a time. This may be desirable if a buffer can only be accessed with long instructions, as in certain long-word-wide memory-mapped peripherals. The source and destination must be long-aligned.

**RETURNS** N/A

**SEE ALSO** `bLib`, `bcopy()`

---

## bcopyWords()

<b>NAME</b>	<b>bcopyWords()</b> – copy one buffer to another one word at a time
<b>SYNOPSIS</b>	<pre>void bcopyWords (     char * source,           /* pointer to source buffer */     char * destination,     /* pointer to destination buffer */     int   nwords            /* number of words to copy */ )</pre>
<b>DESCRIPTION</b>	This routine copies the first <i>nwords</i> words from <i>source</i> to <i>destination</i> one word at a time. This may be desirable if a buffer can only be accessed with word instructions, as in certain word-wide memory-mapped peripherals. The source and destination must be word-aligned.
<b>RETURNS</b>	N/A
<b>SEE ALSO</b>	<b>bLib</b> , <b>bcopy()</b>

---

## bd()

<b>NAME</b>	<b>bd()</b> – delete a breakpoint
<b>SYNOPSIS</b>	<pre>STATUS bd (     INSTR * addr,           /* address of breakpoint to delete */     int   task             /* task for which to delete breakpoint, 0 = */                           /* delete for all tasks */ )</pre>
<b>DESCRIPTION</b>	<p>This routine deletes a specified breakpoint.</p> <p>To execute, enter:</p> <pre>-&gt; bd addr [,task]</pre> <p>If <i>task</i> is omitted or zero, the breakpoint will be removed for all tasks. If the breakpoint applies to all tasks, removing it for only a single task will be ineffective. It must be removed for all tasks and then set for just those tasks desired. Temporary breakpoints inserted by the routines <b>so()</b> or <b>cret()</b> can also be deleted.</p>

**bdall()**

- RETURNS** OK, or **ERROR** if there is no breakpoint at the specified address.
- SEE ALSO** **dbgLib**, **b()**, *VxWorks Programmer's Guide: Target Shell*, **windsh**, *Tornado User's Guide: Shell*

**bdall()**

**NAME** **bdall()** – delete all breakpoints

**SYNOPSIS**

```
STATUS bdall
(
    int task                /* task for which to delete breakpoints, 0 */
                           /* = delete for all tasks */
)
```

**DESCRIPTION** This routine removes all breakpoints.

To execute, enter:

```
-> bdall [task]
```

If *task* is specified, all breakpoints that apply to that task are removed. If *task* is omitted, all breakpoints for all tasks are removed. Temporary breakpoints inserted by **so()** or **cret()** are not deleted; use **bd()** instead.

**RETURNS** OK, always.

**SEE ALSO** **dbgLib**, **bd()**, *VxWorks Programmer's Guide: Target Shell*, **windsh**, *Tornado User's Guide: Shell*

**bfill()**

**NAME** **bfill()** – fill a buffer with a specified character

**SYNOPSIS**

```
void bfill
(
    char * buf,            /* pointer to buffer */
    int  nbytes,          /* number of bytes to fill */
    int  ch                /* char with which to fill buffer */
)
```

<b>DESCRIPTION</b>	This routine fills the first <i>nbytes</i> characters of a buffer with the character <i>ch</i> . Filling is done in the most efficient way possible, which may be long-word, or even multiple-long-word stores, on some architectures. In general, the fill will be significantly faster if the buffer is long-word aligned. (For filling that is restricted to byte stores, see the manual entry for <code>bfillBytes()</code> .)
<b>RETURNS</b>	N/A
<b>SEE ALSO</b>	<code>bLib</code> , <code>bfillBytes()</code>

---

## bfillBytes( )

**NAME** `bfillBytes()` – fill buffer with a specified character one byte at a time

**SYNOPSIS**

```
void bfillBytes
(
    char * buf,           /* pointer to buffer */
    int   nbytes,       /* number of bytes to fill */
    int   ch             /* char with which to fill buffer */
)
```

<b>DESCRIPTION</b>	This routine fills the first <i>nbytes</i> characters of the specified buffer with the character <i>ch</i> one byte at a time. This may be desirable if a buffer can only be accessed with byte instructions, as in certain byte-wide memory-mapped peripherals.
<b>RETURNS</b>	N/A
<b>SEE ALSO</b>	<code>bLib</code> , <code>bfill()</code>

---

## bh( )

**NAME** `bh()` – set a hardware breakpoint

**SYNOPSIS**

```
STATUS bh
(
    INSTR * addr,        /* where to set breakpoint, or 0 = display */
                        /* all breakpoints */
    int   access,       /* access type (arch dependant) */
    int   task,         /* task for which to set breakpoint, 0 = */
)
```

**bind()**

```

                                /* set all tasks */
    int      count,              /* number of passes before hit */
    BOOL     quiet              /* TRUE = don't print debugging info, FALSE */
                                /* = print debugging info */
)

```

**DESCRIPTION** This routine is used to set a hardware breakpoint. If the architecture allows it, this function will add the breakpoint to the list of breakpoints and set the hardware breakpoint register(s). For more information, see the manual entry for **b()**.

---

**NOTE:** The types of hardware breakpoints vary with the architectures. Generally, a hardware breakpoint can be a data breakpoint or an instruction breakpoint.

---

**RETURNS** OK, or ERROR if *addr* is illegal or the hardware breakpoint table is full.

**SEE ALSO** **dbgLib**, **b()**, *VxWorks Programmer's Guide: Target Shell*

---

## bind()

**NAME** **bind()** – bind a name to a socket

**SYNOPSIS**

```

STATUS bind
(
    int          s,          /* socket descriptor */
    struct sockaddr * name, /* name to be bound */
    int         namelen /* length of name */
)

```

**DESCRIPTION** This routine associates a network address (also referred to as its “name”) with a specified socket so that other processes can connect or send to it. When a socket is created with **socket()**, it belongs to an address family but has no assigned name.

**RETURNS** OK, or ERROR if there is an invalid socket, the address is either unavailable or in use, or the socket is already bound.

**SEE ALSO** **sockLib**

---

## bindresvport()

<b>NAME</b>	<b>bindresvport()</b> – bind a socket to a privileged IP port
<b>SYNOPSIS</b>	<pre><b>STATUS</b> bindresvport (     int          sd, /* socket to be bound */     struct sockaddr_in * sin /* socket address -- value/result */ )</pre>
<b>DESCRIPTION</b>	This routine picks a port number between 600 and 1023 that is not being used by any other programs and binds the socket passed as <i>sd</i> to that port. Privileged IP ports (numbers between and including 0 and 1023) are reserved for privileged programs.
<b>RETURNS</b>	OK, or <b>ERROR</b> if the address family specified in <i>sin</i> is not supported or the call fails.
<b>SEE ALSO</b>	<b>remLib</b>

---

## binvert()

<b>NAME</b>	<b>binvert()</b> – invert the order of bytes in a buffer
<b>SYNOPSIS</b>	<pre>void binvert (     char * buf,          /* pointer to buffer to invert */     int  nbytes         /* number of bytes in buffer */ )</pre>
<b>DESCRIPTION</b>	This routine inverts an entire buffer, byte by byte. For example, the buffer {1, 2, 3, 4, 5} would become {5, 4, 3, 2, 1}.
<b>RETURNS</b>	N/A
<b>SEE ALSO</b>	<b>bLib</b>

---

## bootBpAnchorExtract()

<b>NAME</b>	<b>bootBpAnchorExtract()</b> – extract a backplane address from a device field
<b>SYNOPSIS</b>	<pre>STATUS bootBpAnchorExtract (     char * string,          /* string containing adrs field */     char * *pAnchorAdrs   /* pointer where to return anchor address */ )</pre>
<b>DESCRIPTION</b>	<p>This routine extracts the optional backplane anchor address field from a boot device field. The anchor can be specified for the backplane driver by appending to the device name (<i>i.e.</i>, “bp”) an equal sign (=) and the address in hexadecimal. For example, the “boot device” field of the boot parameters could be specified as:</p> <pre>boot device: bp=800000</pre> <p>In this case, the backplane anchor address would be at address 0x800000, instead of the default specified in <b>config.h</b>.</p> <p>This routine picks off the optional trailing anchor address by replacing the equal sign (=) in the specified string with an EOS and then scanning the remainder as a hex number. This number, the anchor address, is returned via the <i>pAnchorAdrs</i> pointer.</p>
<b>RETURNS</b>	1 if the anchor address in <i>string</i> is specified correctly, 0 if the anchor address in <i>string</i> is not specified, or -1 if an invalid anchor address is specified in <i>string</i> .
<b>SEE ALSO</b>	<b>bootLib</b>

---

## bootChange()

<b>NAME</b>	<b>bootChange()</b> – change the boot line
<b>SYNOPSIS</b>	<pre>void bootChange (void)</pre>
<b>DESCRIPTION</b>	<p>This command changes the boot line used in the boot ROMs. This is useful during a remote login session. After changing the boot parameters, you can reboot the target with the <b>reboot()</b> command, and then terminate your login (~.) and remotely log in again. As soon as the system has rebooted, you will be logged in again.</p> <p>This command stores the new boot line in non-volatile RAM, if the target has it.</p>

**RETURNS** N/A

**SEE ALSO** `usrLib`, `windsh`, *Tornado User's Guide: Shell*

**B**

---

## bootLeaseExtract()

**NAME** `bootLeaseExtract()` – extract the lease information from an Internet address

**SYNOPSIS**

```
int bootLeaseExtract
(
    char * string,          /* string containing addr field */
    u_long * pLeaseLen,    /* pointer to storage for lease duration */
    u_long * pLeaseStart   /* pointer to storage for lease origin */
)
```

**DESCRIPTION** This routine extracts the optional lease duration and lease origin fields from an Internet address field for use with DHCP. The lease duration can be specified by appending a colon and the lease duration to the netmask field. For example, the “inet on ethernet” field of the boot parameters could be specified as:

```
inet on ethernet: 90.1.0.1:ffff0000:1000
```

If no netmask is specified, the contents of the field could be:

```
inet on ethernet: 90.1.0.1::ffffffff
```

In the first case, the lease duration for the address is 1000 seconds. The second case indicates an infinite lease, and does not specify a netmask for the address. At the beginning of the boot process, the value of the lease duration field is used to specify the requested lease duration. If the field not included, the value of `DHCP_DEFAULT_LEASE` is used instead.

The lease origin is specified with the same format as the lease duration, but is added during the boot process. The presence of the lease origin field distinguishes addresses assigned by a DHCP server from addresses entered manually. Addresses assigned by a DHCP server may be replaced if the bootstrap loader uses DHCP to obtain configuration parameters. The value of the lease origin field at the beginning of the boot process is ignored.

This routine extracts the optional lease duration by replacing the preceding colon in the specified string with an EOS and then scanning the remainder as a number. The lease duration and lease origin values are returned via the `pLeaseLen` and `pLeaseStart` pointers, if those parameters are not NULL.

**RETURNS** 2 if both lease values are specified correctly in *string*, or  
-2 if one of the two values is specified incorrectly.

If only the lease duration is found, it returns:  
1 if the lease duration in *string* is specified correctly,  
0 if the lease duration is not specified in *string*, or  
-1 if an invalid lease duration is specified in *string*.

**SEE ALSO** **bootLib**

---

## bootNetmaskExtract()

**NAME** **bootNetmaskExtract()** – extract the net mask field from an Internet address

**SYNOPSIS**

```
STATUS bootNetmaskExtract  
(  
    char * string,          /* string containing addr field */  
    int * pNetmask         /* pointer where to return net mask */  
)
```

**DESCRIPTION** This routine extracts the optional subnet mask field from an Internet address field. Subnet masks can be specified for an Internet interface by appending to the Internet address a colon and the net mask in hexadecimal. For example, the “inet on ethernet” field of the boot parameters could be specified as:

```
inet on ethernet: 90.1.0.1:ffff0000
```

In this case, the network portion of the address (normally just 90) is extended by the subnet mask (to 90.1). This routine extracts the optional trailing subnet mask by replacing the colon in the specified string with an EOS and then scanning the remainder as a hex number. This number, the net mask, is returned via the *pNetmask* pointer.

This routine also handles an empty netmask field used as a placeholder for the lease duration field (see **bootLeaseExtract()**). In that case, the colon separator is replaced with an EOS and the value of netmask is set to 0.

**RETURNS** 1 if the subnet mask in *string* is specified correctly,  
0 if the subnet mask in *string* is not specified, or  
-1 if an invalid subnet mask is specified in *string*.

**SEE ALSO** **bootLib**

---

## bootParamsPrompt()

<b>NAME</b>	<b>bootParamsPrompt()</b> – prompt for boot line parameters
<b>SYNOPSIS</b>	<pre>void bootParamsPrompt (     char * string          /* default boot line */ )</pre>
<b>DESCRIPTION</b>	<p>This routine displays the current value of each boot parameter and prompts the user for a new value. Typing a RETURN leaves the parameter unchanged. Typing a period (.) clears the parameter.</p> <p>The parameter <i>string</i> holds the initial values. The new boot line is copied over <i>string</i>. If there are no initial values, <i>string</i> is empty on entry.</p>
<b>RETURNS</b>	N/A
<b>SEE ALSO</b>	<b>bootLib</b>

---

## bootParamsShow()

<b>NAME</b>	<b>bootParamsShow()</b> – display boot line parameters
<b>SYNOPSIS</b>	<pre>void bootParamsShow (     char * paramString    /* boot parameter string */ )</pre>
<b>DESCRIPTION</b>	<p>This routine displays the boot parameters in the specified boot string one parameter per line.</p>
<b>RETURNS</b>	N/A
<b>SEE ALSO</b>	<b>bootLib</b>

---

## bootpLibInit()

<b>NAME</b>	<b>bootpLibInit()</b> – BOOTP client library initialization
<b>SYNOPSIS</b>	<pre><b>STATUS</b> bootpLibInit (     int maxSize           /* largest link-level header, in bytes */ )</pre>
<b>DESCRIPTION</b>	<p>This routine creates and initializes the global data structures used by the BOOTP client library to obtain configuration parameters. The <i>maxSize</i> parameter specifies the largest link level header for all supported devices. This value determines the maximum length of the outgoing IP packet containing a BOOTP message.</p> <p>This routine must be called before using any other library routines. The routine is called automatically if <code>INCLUDE_BOOTP</code> is defined at the time the system is built and uses the <code>BOOTP_MAXSIZE</code> configuration setting for the <i>maxSize</i> parameter.</p>
<b>RETURNS</b>	OK, or ERROR if initialization fails.
<b>ERRNO</b>	S_bootpLib_MEM_ERROR
<b>SEE ALSO</b>	bootpLib

---

## bootpMsgGet()

<b>NAME</b>	<b>bootpMsgGet()</b> – send a BOOTP request message and retrieve reply
<b>SYNOPSIS</b>	<pre><b>STATUS</b> bootpMsgGet (     struct ifnet *  pIf,           /* network device for message exchange */     struct in_addr * pIpDest,     /* destination IP address for request */     USHORT         srcPort,       /* UDP source port for request */     USHORT         dstPort,       /* UDP destination port for request */     BOOTP_MSG *    pBootpMsg,     /* request template and reply storage */     u_int          maxSends       /* maximum number of transmit attempts */ )</pre>
<b>DESCRIPTION</b>	<p>This routine sends a BOOTP request using the <i>pIf</i> network interface and waits for a reply. <i>pIpDest</i> specifies the destination IP address. It must be equal to either the broadcast address (255.255.255.255) or the IP address of a specific BOOTP server directly reachable using the network interface. The interface must support broadcasting in the first case.</p>

The *srcPort* and *dstPort* arguments support sending and receiving BOOTP messages with arbitrary UDP ports. To receive replies, any BOOTP server must send those responses to the source port from the request. To comply with the RFC 1542 clarification, the request message must be sent to the reserved BOOTP server port (67) using the reserved BOOTP client port (68).

Except for the UDP port numbers, this routine only sets the **bp\_xid** and **bp\_secs** fields in the outgoing BOOTP message. All other fields in that message use the values from the *pBootpMsg* argument, which later holds the contents of any BOOTP reply received.

The *maxSends* parameter specifies the total number of requests to transmit if no reply is received. The retransmission interval starts at 4 seconds and doubles with each attempt up to a maximum of 64 seconds. Any subsequent retransmissions will occur at that maximum interval. To reduce the chances of network flooding, the timeout interval before each retransmission includes a randomized delay of plus or minus one second from the base value. After the final transmission, this routine will wait for the current interval to expire before returning a timeout error.

---

**NOTE:** The target must be able to respond to an ARP request for any IP address specified in the request template's **bp\_ciaddr** field.

---

**RETURNS** OK, or ERROR.

**ERRNO** S\_bootpLib\_INVALID\_ARGUMENT  
S\_bootpLib\_NO\_BROADCASTS  
S\_bootpLib\_TIME\_OUT

**SEE ALSO** bootpLib

---

## bootpParamsGet()

**NAME** bootpParamsGet() – retrieve boot parameters using BOOTP

**SYNOPSIS**

```
STATUS bootpParamsGet
(
    struct ifnet *    pIf,          /* network device used by client */
    u_int            maxSends,     /* maximum transmit attempts */
    struct in_addr * pClientAddr,  /* retrieved client address buffer */
    struct in_addr * pServerAddr,  /* buffer for server's IP address */
    char *           pHostName,   /* 64 byte (max) host name buffer */
    char *           pBootFile,   /* 128 byte (max) file name buffer */
    struct bootpParams * pBootpParams /* parameters descriptor */
)
```

**DESCRIPTION**

This routine performs a BOOTP message exchange according to the process described in RFC 1542, so the server and client UDP ports are always equal to the defined values of 67 and 68.

The *pIf* argument indicates the network device which will be used to send and receive BOOTP messages. The BOOTP client only supports devices attached to the IP protocol with the MUX/END interface. The MTU size must be large enough to receive an IP packet of 328 bytes (corresponding to the BOOTP message length of 300 bytes). The specified device also must be capable of sending broadcast messages, unless this routine sends the request messages directly to the IP address of a specific server.

The *maxSends* parameter specifies the total number of requests before this routine stops waiting for a reply. After the final request, this routine will wait for the current interval before returning error. The timeout interval following each request follows RFC 1542, beginning at 4 seconds and doubling until a maximum limit of 64 seconds.

The *pClientAddr* parameter provides optional storage for the assigned IP address from the **yiaddr** field of a BOOTP reply. Since this routine can execute before the system is capable of accepting unicast datagrams or responding to ARP requests for a specific IP address, the corresponding **ciaddr** field in the BOOTP request message is equal to zero.

The *pServerAddr* parameter provides optional storage for the IP address of the responding server (from the **siaddr** field of a BOOTP reply). This routine broadcasts the BOOTP request message unless this buffer is available (*i.e.*, not NULL) and contains the explicit IP address of a BOOTP server as a non-zero value.

The *pHostName* parameter provides optional storage for the server's host name (from the **sname** field of a BOOTP reply). This routine also copies any initial string in that buffer into the **sname** field of the BOOTP request (which restricts booting to a specified host).

The *pBootFile* parameter provides optional storage for the boot file name (from the **file** field of a BOOTP reply). This routine also copies any initial string in that buffer into the **file** field of the BOOTP request message, which typically supplies a generic name to the server.

The remaining fields in the BOOTP request message use the values which RFC 1542 defines. In particular, the **giaddr** field is set to zero and the suggested "magic cookie" is always inserted in the (otherwise empty) **vend** field.

The *pBootpParams* argument provides access to any options defined in RFC 1533 using the following definition:

```
struct bootpParams
{
    struct in_addr *      netmask;
    unsigned short *    timeOffset;
    struct in_addr_list * routers;
    struct in_addr_list * timeServers;
    struct in_addr_list * nameServers;
    struct in_addr_list * dnsServers;
```

```

struct in_addr_list *   logServers;
struct in_addr_list *   cookieServers;
struct in_addr_list *   lprServers;
struct in_addr_list *   impressServers;
struct in_addr_list *   rlpServers;
char *                  clientName;
unsigned short *        filesize;
char *                  dumpfile;
char *                  domainName;
struct in_addr *        swapServer;
char *                  rootPath;
char *                  extoptPath;
unsigned char *         ipForward;
unsigned char *         nonlocalSourceRoute;
struct in_addr_list *   policyFilter;
unsigned short *        maxDgramSize;
unsigned char *         ipTTL;
unsigned long *         mtuTimeout;
struct ushort_list *    mtuTable;
unsigned short *        interfaceMTU;
unsigned char *         allSubnetsLocal;
struct in_addr *        broadcastAddr;
unsigned char *         maskDiscover;
unsigned char *         maskSupplier;
unsigned char *         routerDiscover;
struct in_addr *        routerDiscAddr;
struct in_addr_list *   staticRoutes;
unsigned char *         arpTrailers;
unsigned long *         arpTimeout;
unsigned char *         etherPacketType;
unsigned char *         tcpTTL;
unsigned long *         tcpInterval;
unsigned char *         tcpGarbage;
char *                  nisDomain;
struct in_addr_list *   nisServers;
struct in_addr_list *   ntpServers;
char *                  vendString;
struct in_addr_list *   nbnServers;
struct in_addr_list *   nbddServers;
unsigned char *         nbNodeType;
char *                  nbScope;
struct in_addr_list *   xFontServers;
struct in_addr_list *   xDisplayManagers;
char *                  nispDomain;
struct in_addr_list *   nispServers;
struct in_addr_list *   ipAgents;

```

```
struct in_addr_list *    smtpServers;  
struct in_addr_list *    pop3Servers;  
struct in_addr_list *    nntpServers;  
struct in_addr_list *    wwwServers;  
struct in_addr_list *    fingerServers;  
struct in_addr_list *    ircServers;  
struct in_addr_list *    stServers;  
struct in_addr_list *    stdaServers;  
};
```

This structure allows the retrieval of any BOOTP option specified in RFC 1533. The list of 2-byte (unsigned short) values is defined as:

```
struct ushort_list  
{  
    unsigned char    num;  
    unsigned short * shortlist;  
};
```

The IP address lists use the following similar definition:

```
struct in_addr_list  
{  
    unsigned char    num;  
    struct in_addr * addrlist;  
};
```

When these lists are present, the routine stores values retrieved from the BOOTP reply in the location indicated by the **shortlist** or **addrlist** members. The amount of space available is indicated by the **num** member. When the routine returns, the **num** member indicates the actual number of entries retrieved. In the case of **bootpParams.policyFilter.num** and **bootpParams.staticRoutes.num**, the **num** member value should be interpreted as the number of IP address pairs requested and received.

The contents of the BOOTP parameter descriptor implicitly selects options for retrieval from the BOOTP server. This routine attempts to retrieve the values for any options whose corresponding field pointers are non-NULL values. To obtain these parameters, the BOOTP server must support the vendor-specific options described in RFC 1048 (or its successors) and the corresponding parameters must be specified in the BOOTP server database. Where meaningful, the values are returned in host byte order.

The BOOTP request issued during system startup with this routine attempts to retrieve a subnet mask for the boot device, in addition to the host and client addresses and the boot file name.

**RETURNS** OK, or **ERROR** if unsuccessful.

**SEE ALSO** **bootpLib**, **bootLib**, RFC 1048, RFC 1533

---

## bootStringToStruct()

- NAME** `bootStringToStruct()` – interpret the boot parameters from the boot line
- SYNOPSIS**
- ```
char *bootStringToStruct
(
    char *      bootString, /* boot line to be parsed */
    BOOT_PARAMS * pBootParams /* where to return parsed boot line */
)
```
- DESCRIPTION** This routine parses the ASCII string and returns the values into the provided parameters. For a description of the format of the boot line, see the manual entry for **bootLib**
- RETURNS** A pointer to the last character successfully parsed plus one (points to EOS, if OK). The entire boot line is parsed.
- SEE ALSO** `bootLib`

---

## bootStructToString()

- NAME** `bootStructToString()` – construct a boot line
- SYNOPSIS**
- ```
STATUS bootStructToString
(
    char *      paramString, /* where to return the encoded boot line */
    BOOT_PARAMS * pBootParams /* boot line structure to be encoded */
)
```
- DESCRIPTION** This routine encodes a boot line using the specified boot parameters. For a description of the format of the boot line, see the manual entry for **bootLib**.
- RETURNS** OK.
- SEE ALSO** `bootLib`

## bpfDevCreate()

<b>NAME</b>	<b>bpfDevCreate()</b> – create Berkeley Packet Filter device
<b>SYNOPSIS</b>	<pre><b>STATUS</b> bpfDevCreate (     char * pDevName,          /* I/O system device name */     int   numUnits,          /* number of device units */     int   bufSize            /* BPF device block size (0 for default) */ )</pre>
<b>DESCRIPTION</b>	This routine creates a Berkeley Packet Filter device. Each of the <i>numUnits</i> units corresponds to a single available file descriptor for monitoring a network device. The <i>pDevName</i> parameter provides the name of the BPF device to the I/O system. The default name of <code>"/bpf"</code> (assigned if <i>pDevName</i> is NULL) produces units named <code>"/bpf0"</code> , <code>"/bpf1"</code> , etc., up to the <i>numUnits</i> limit.
<b>RETURNS</b>	OK, or ERROR if device creation failed.
<b>ERRNO</b>	S_ioLib_NO_DRIVER
<b>SEE ALSO</b>	bpfDrv

---

## bpfDevDelete()

<b>NAME</b>	<b>bpfDevDelete()</b> – destroy Berkeley Packet Filter device
<b>SYNOPSIS</b>	<pre><b>STATUS</b> bpfDevDelete (     char * pDevName          /* name of BPF device to remove */ )</pre>
<b>DESCRIPTION</b>	This routine removes a Berkeley Packet Filter device and releases all allocated memory. It will close any open files using the device.
<b>RETURNS</b>	OK, or ERROR if device not found
<b>ERRNO</b>	S_ioLib_NO_DRIVER
<b>SEE ALSO</b>	bpfDrv

---

## bpfDrv()

<b>NAME</b>	<b>bpfDrv()</b> – initialize the BPF driver
<b>SYNOPSIS</b>	<b>STATUS</b> <b>bpfDrv</b> (void)
<b>DESCRIPTION</b>	This routine installs the Berkeley Packet Filter driver for access through the I/O system. It is required before performing any I/O operations and is executed automatically if <b>INCLUDE_BPF</b> is defined at the time the system is built. Subsequent calls to the routine just count the number of users with BPF access.
<b>RETURNS</b>	OK, or <b>ERROR</b> if initialization fails.
<b>ERRNO</b>	N/A
<b>SEE ALSO</b>	<b>bpfDrv</b>

---

## bsearch()

<b>NAME</b>	<b>bsearch()</b> – perform a binary search (ANSI)
<b>SYNOPSIS</b>	<pre>void * bsearch (     const void * key,          /* element to match */     const void * base0,       /* initial element in array */     size_t      nmemb,        /* array to search */     size_t      size,         /* size of array element */     int (* compar) (const void * , const void * )                                /* comparison function */ )</pre>
<b>DESCRIPTION</b>	<p>This routine searches an array of <i>nmemb</i> objects, the initial element of which is pointed to by <i>base0</i>, for an element that matches the object pointed to by <i>key</i>. The <i>size</i> of each element of the array is specified by <i>size</i>.</p> <p>The comparison function pointed to by <i>compar</i> is called with two arguments that point to the <i>key</i> object and to an array element, in that order. The function shall return an integer less than, equal to, or greater than zero if the <i>key</i> object is considered, respectively, to be less than, to match, or to be greater than the array element. The array shall consist of all the elements that compare greater than the <i>key</i> object, in that order.</p>

<b>INCLUDE FILES</b>	<b>stdlib.h</b>
<b>RETURNS</b>	A pointer to a matching element of the array, or a NULL pointer if no match is found. If two elements compare as equal, which element is matched is unspecified.
<b>SEE ALSO</b>	<b>ansiStdlib</b>

---

## **bswap()**

**NAME** **bswap()** – swap buffers

**SYNOPSIS**

```
void bswap
(
    char * buf1,          /* pointer to first buffer */
    char * buf2,          /* pointer to second buffer */
    int   nbytes         /* number of bytes to swap */
)
```

**DESCRIPTION** This routine exchanges the first *nbytes* of the two specified buffers.

**RETURNS** N/A

**SEE ALSO** **bLib**

---

## **bzero()**

**NAME** **bzero()** – zero out a buffer

**SYNOPSIS**

```
void bzero
(
    char * buffer,       /* buffer to be zeroed */
    int   nbytes        /* number of bytes in buffer */
)
```

**DESCRIPTION** This routine fills the first *nbytes* characters of the specified buffer with 0.

**RETURNS** N/A

**SEE ALSO** **bLib**

---

## c()

**NAME** c() – continue from a breakpoint

**SYNOPSIS**

```
STATUS c
(
    int    task,    /* task that should proceed from breakpoint */
    INSTR * addr,  /* address to continue at; 0 = next instruction */
)
```

**DESCRIPTION** This routine continues the execution of a task that has stopped at a breakpoint. To execute, enter:

```
-> c [task [,addr[,addr1]]]
```

If *task* is omitted or zero, the last task referenced is assumed. If *addr* is non-zero, the program counter is changed to *addr*; if *addr1* is non-zero, the next program counter is changed to *addr1*, and the task is continued.

---

**WARNING:** When a task is continued, `c()` does not distinguish between a suspended task or a task suspended by the debugger. Therefore, its use should be restricted to only those tasks being debugged.

---

**RETURNS** OK, or ERROR if the specified task does not exist.

**SEE ALSO** `dbgLib`, `tr()`, *VxWorks Programmer's Guide: Target Shell*, `windsh`, *Tornado User's Guide: Shell*

---

## cache4kcLibInit()

**NAME** cache4kcLibInit() – initialize the 4kc cache library

**SYNOPSIS**

```
STATUS cache4kcLibInit
(
    CACHE_MODE instMode,    /* instruction cache mode */
    CACHE_MODE dataMode,   /* data cache mode */
    UINT32      iCacheSize,
    UINT32      iCacheLineSize,
    UINT32      dCacheSize,
    UINT32      dCacheLineSize
)
```

**cacheArchClearEntry()**

- DESCRIPTION** This routine initializes the function pointers for the 4kc cache library. The board support package can select this cache library by assigning the function pointer *sysCacheLibInit* to **cache4kcLibInit()**.
- RETURNS** OK.
- SEE ALSO** **cache4kcLib**

---

## cacheArchClearEntry()

**NAME** **cacheArchClearEntry()** – clear an entry from a cache (68K, x86)

**SYNOPSIS**

```
STATUS cacheArchClearEntry
(
    CACHE_TYPE cache,          /* cache to clear entry for */
    void *    address        /* entry to clear */
)
```

**DESCRIPTION** This routine clears a specified entry from the specified cache.

For 68040 processors, this routine clears the cache line from the cache in which the cache entry resides.

For the MC68060 processor, when the instruction cache is cleared (invalidated) the branch cache is also invalidated by the hardware. One line in the branch cache cannot be invalidated so each time the branch cache is entirely invalidated.

For 386 family processors do not have a cache, thus it does nothing. The 486, P5(Pentium), and P6(PentiumPro, II, III) family processors do have a cache but does not support a line by line cache control, thus it performs WBINVD instruction. The P7(Pentium4) family processors support the line by line cache control with CLFLUSH instruction, thus flushes the specified cache line.

**RETURNS** OK, or ERROR if the cache type is invalid or the cache control is not supported.

**SEE ALSO** **cacheArchLib**

## cacheArchLibInit()

**NAME** cacheArchLibInit() – initialize the cache library

**SYNOPSIS**

```
STATUS cacheArchLibInit
(
    CACHE_MODE instMode,      /* instruction cache mode */
    CACHE_MODE dataMode     /* data cache mode */
)
```

**DESCRIPTION** This routine initializes the cache library for the following processor cache families: Motorola 68K, Intel 960, Intel x86, PowerPC ARM, and the Solaris and Windows simulators. It initializes the function pointers and configures the caches to the specified cache modes.

**68K PROCESSORS** The caching modes vary for members of the 68K processor family:

<b>68020</b>	<b>CACHE_WRITETHROUGH</b>	(instruction cache only)
<b>68030</b>	<b>CACHE_WRITETHROUGH</b>	
	<b>CACHE_BURST_ENABLE</b>	
	<b>CACHE_BURST_DISABLE</b>	
	<b>CACHE_WRITEALLOCATE</b>	(data cache only)
	<b>CACHE_NO_WRITEALLOCATE</b>	(data cache only)
<b>68040</b>	<b>CACHE_WRITETHROUGH</b>	
	<b>CACHE_COPYBACK</b>	(data cache only)
	<b>CACHE_INH_SERIAL</b>	(data cache only)
	<b>CACHE_INH_NONSERIAL</b>	(data cache only)
	<b>CACHE_BURST_ENABLE</b>	(data cache only)
	<b>CACHE_NO_WRITEALLOCATE</b>	(data cache only)
<b>68060</b>	<b>CACHE_WRITETHROUGH</b>	
	<b>CACHE_COPYBACK</b>	(data cache only)
	<b>CACHE_INH_PRECISE</b>	(data cache only)
	<b>CACHE_INH_IMPRECISE</b>	(data cache only)
	<b>CACHE_BURST_ENABLE</b>	(data cache only)

The write-through, copy-back, serial, non-serial, precise and non precise modes change the state of the data transparent translation register (DTTR0) CM bits. Only DTTR0 is modified, since it typically maps DRAM space.

**x86 PROCESSORS** The caching mode **CACHE\_WRITETHROUGH** is available for the 486 family processors. The caching mode **CACHE\_COPYBACK** becomes available for the P5(Pentium) family

processors. The caching mode (`CACHE_COPYBACK` | `CACHE_SNOOP_ENABLE`) becomes available for the P6(PentiumPro, II, III) and P7(Pentium4) family processors.

#### POWER PC PROCESSORS

Modes should be set before caching is enabled. If two contradictory flags are set (for example, enable/disable), no action is taken for any of the input flags.

#### ARM PROCESSORS

The caching capabilities and modes vary for members of the ARM processor family. All caches are provided on-chip, so cache support is mostly an architecture issue, not a BSP issue. However, the memory map is BSP-specific and some functions need knowledge of the memory map, so they have to be provided in the BSP.

ARM7TDMI (In ARM or Thumb state)

No cache or MMU at all. Dummy routine provided, so that `INCLUDE_CACHE_SUPPORT` can be defined (the default BSP configuration).

ARM710A

Combined instruction and data cache. Actually a write-through cache, but separate write-buffer effectively makes this a copy-back cache if the write-buffer is enabled. Use write-through/copy-back argument to decide whether to enable write buffer. Data and instruction cache modes must be identical.

ARM810

Combined instruction and data cache. Write-through and copy-back cache modes, but separate write-buffer effectively makes even write-through a copy-back cache as all writes are buffered, when cache is enabled. Data and instruction cache modes must be identical.

ARMSA110

Separate instruction and data caches. Write-through and copy-back cache mode for data, but separate write-buffer effectively makes even write-through a copy-back cache as all writes are buffered, when cache is enabled.

**RETURNS**            **OK**

**SEE ALSO**            **cacheArchLib**

---

## cacheAuLibInit()

**NAME** cacheAuLibInit() – initialize the Au cache library

**SYNOPSIS**

```
STATUS cacheAuLibInit
(
    CACHE_MODE instMode,      /* instruction cache mode */
    CACHE_MODE dataMode,     /* data cache mode */
    UINT32      iCacheSize,
    UINT32      iCacheLineSize,
    UINT32      dCacheSize,
    UINT32      dCacheLineSize
)
```

**DESCRIPTION** This routine initializes the function pointers for the Au cache library. The board support package can select this cache library by assigning the function pointer *sysCacheLibInit* to *cacheAuLibInit()*.

**RETURNS** OK.

**SEE ALSO** cacheAuLib

---

## cacheClear()

**NAME** cacheClear() – clear all or some entries from a cache

**SYNOPSIS**

```
STATUS cacheClear
(
    CACHE_TYPE cache,        /* cache to clear */
    void *     address,     /* virtual address */
    size_t     bytes       /* number of bytes to clear */
)
```

**DESCRIPTION** This routine flushes and invalidates all or some entries in the specified cache.

**RETURNS** OK, or ERROR if the cache type is invalid or the cache control is not supported.

**SEE ALSO** cacheLib

## cacheCy604ClearLine()

**NAME** cacheCy604ClearLine() – clear a line from a CY7C604 cache

**SYNOPSIS**

```
STATUS cacheCy604ClearLine
(
    CACHE_TYPE cache,      /* cache to clear */
    void *      address    /* virtual address */
)
```

**DESCRIPTION** This routine flushes and invalidates a specified line from the specified CY7C604 cache.

**RETURNS** OK, or ERROR if the cache type is invalid or the cache control is not supported.

**SEE ALSO** cacheCy604Lib

---

## cacheCy604ClearPage()

**NAME** cacheCy604ClearPage() – clear a page from a CY7C604 cache

**SYNOPSIS**

```
STATUS cacheCy604ClearPage
(
    CACHE_TYPE cache,      /* cache to clear */
    void *      address    /* virtual address */
)
```

**DESCRIPTION** This routine flushes and invalidates the specified page from the specified CY7C604 cache.

**RETURNS** OK, or ERROR if the cache type is invalid or the cache control is not supported.

**SEE ALSO** cacheCy604Lib

---

## cacheCy604ClearRegion()

**NAME** cacheCy604ClearRegion() – clear a region from a CY7C604 cache

**SYNOPSIS**

```
STATUS cacheCy604ClearRegion
(
    CACHE_TYPE cache,          /* cache to clear */
    void *      address       /* virtual address */
)
```

**DESCRIPTION** This routine flushes and invalidates a specified region from the specified CY7C604 cache.

**RETURNS** OK, or ERROR if the cache type is invalid or the cache control is not supported.

**SEE ALSO** cacheCy604Lib

---

## cacheCy604ClearSegment()

**NAME** cacheCy604ClearSegment() – clear a segment from a CY7C604 cache

**SYNOPSIS**

```
STATUS cacheCy604ClearSegment
(
    CACHE_TYPE cache,          /* cache to clear */
    void *      address       /* virtual address */
)
```

**DESCRIPTION** This routine flushes and invalidates a specified segment from the specified CY7C604 cache.

**RETURNS** OK, or ERROR if the cache type is invalid or the cache control is not supported.

**SEE ALSO** cacheCy604Lib

## cacheCy604LibInit()

**NAME** cacheCy604LibInit() – initialize the Cypress CY7C604 cache library

**SYNOPSIS**

```
STATUS cacheCy604LibInit
(
    CACHE_MODE instMode,    /* instruction cache mode */
    CACHE_MODE dataMode    /* data cache mode */
)
```

**DESCRIPTION** This routine initializes the function pointers for the Cypress CY7C604 cache library. The board support package can select this cache library by assigning the function pointer `sysCacheLibInit` to `cacheCy604LibInit()`.

The available cache modes are `CACHE_WRITETHROUGH` and `CACHE_COPYBACK`. Write-through uses “no-write allocate”; copyback uses “write allocate.”

**RETURNS** OK, or ERROR if cache control is not supported.

**SEE ALSO** cacheCy604Lib

---

## cacheDisable()

**NAME** cacheDisable() – disable the specified cache

**SYNOPSIS**

```
STATUS cacheDisable
(
    CACHE_TYPE cache        /* cache to disable */
)
```

**DESCRIPTION** This routine flushes the cache and disables the instruction or data cache.

**RETURNS** OK, or ERROR if the cache type is invalid or the cache control is not supported.

**SEE ALSO** cacheLib

---

## cacheDmaFree()

**NAME** `cacheDmaFree()` – free the buffer acquired with `cacheDmaMalloc()`

**SYNOPSIS**

```
STATUS cacheDmaFree  
(  
    void * pBuf          /* pointer to malloc/free buffer */  
)
```

**DESCRIPTION** This routine frees the buffer returned by `cacheDmaMalloc()`.

**RETURNS** OK, or ERROR if the cache control is not supported.

**SEE ALSO** `cacheLib`

---

## cacheDmaMalloc()

**NAME** `cacheDmaMalloc()` – allocate a cache-safe buffer for DMA devices and drivers

**SYNOPSIS**

```
void * cacheDmaMalloc  
(  
    size_t bytes          /* number of bytes to allocate */  
)
```

**DESCRIPTION** This routine returns a pointer to a section of memory that will not experience any cache coherency problems. Function pointers in the `CACHE_FUNCS` structure provide access to DMA support routines.

**RETURNS** A pointer to the cache-safe buffer, or NULL.

**SEE ALSO** `cacheLib`

## cacheDrvFlush()

**NAME** cacheDrvFlush() – flush the data cache for drivers

**SYNOPSIS**

```
STATUS cacheDrvFlush
(
    CACHE_FUNCS * pFuncs,    /* pointer to CACHE_FUNCS */
    void *      address,    /* virtual address */
    size_t      bytes       /* number of bytes to flush */
)
```

**DESCRIPTION** This routine flushes the data cache entries using the function pointer from the specified set.

**RETURNS** OK, or ERROR if the cache control is not supported.

**SEE ALSO** cacheLib

---

## cacheDrvInvalidate()

**NAME** cacheDrvInvalidate() – invalidate data cache for drivers

**SYNOPSIS**

```
STATUS cacheDrvInvalidate
(
    CACHE_FUNCS * pFuncs,    /* pointer to CACHE_FUNCS */
    void *      address,    /* virtual address */
    size_t      bytes       /* no. of bytes to invalidate */
)
```

**DESCRIPTION** This routine invalidates the data cache entries using the function pointer from the specified set.

**RETURNS** OK, or ERROR if the cache control is not supported.

**SEE ALSO** cacheLib

---

## cacheDrvPhysToVirt()

**NAME** `cacheDrvPhysToVirt()` – translate a physical address for drivers

**SYNOPSIS**

```
void * cacheDrvPhysToVirt
(
    CACHE_FUNCS * pFuncs,      /* pointer to CACHE_FUNCS */
    void *      address        /* physical address */
)
```

**DESCRIPTION** This routine performs a physical-to-virtual address translation using the function pointer from the specified set.

**RETURNS** The virtual address that maps to the physical address argument.

**SEE ALSO** `cacheLib`

---

## cacheDrvVirtToPhys()

**NAME** `cacheDrvVirtToPhys()` – translate a virtual address for drivers

**SYNOPSIS**

```
void * cacheDrvVirtToPhys
(
    CACHE_FUNCS * pFuncs,      /* pointer to CACHE_FUNCS */
    void *      address        /* virtual address */
)
```

**DESCRIPTION** This routine performs a virtual-to-physical address translation using the function pointer from the specified set.

**RETURNS** The physical address translation of a virtual address argument.

**SEE ALSO** `cacheLib`

## cacheEnable()

**NAME** cacheEnable() – enable the specified cache

**SYNOPSIS**

```
STATUS cacheEnable
(
    CACHE_TYPE cache          /* cache to enable */
)
```

**DESCRIPTION** This routine invalidates the cache tags and enables the instruction or data cache.

**RETURNS** OK, or ERROR if the cache type is invalid or the cache control is not supported.

**SEE ALSO** cacheLib

---

## cacheFlush()

**NAME** cacheFlush() – flush all or some of a specified cache

**SYNOPSIS**

```
STATUS cacheFlush
(
    CACHE_TYPE cache,        /* cache to flush */
    void *    address,      /* virtual address */
    size_t    bytes         /* number of bytes to flush */
)
```

**DESCRIPTION** This routine flushes (writes to memory) all or some of the entries in the specified cache. Depending on the cache design, this operation may also invalidate the cache tags. For write-through caches, no work needs to be done since RAM already matches the cached entries. Note that write buffers on the chip may need to be flushed to complete the flush.

**RETURNS** OK, or ERROR if the cache type is invalid or the cache control is not supported.

**SEE ALSO** cacheLib

---

## cacheInvalidate()

**NAME** `cacheInvalidate()` – invalidate all or some of a specified cache

**SYNOPSIS**

```
STATUS cacheInvalidate
(
    CACHE_TYPE cache,          /* cache to invalidate */
    void *      address,      /* virtual address */
    size_t      bytes         /* number of bytes to invalidate */
)
```

**DESCRIPTION** This routine invalidates all or some of the entries in the specified cache. Depending on the cache design, the invalidation may be similar to the flush, or one may invalidate the tags directly.

**RETURNS** OK, or **ERROR** if the cache type is invalid or the cache control is not supported.

**SEE ALSO** `cacheLib`

---

## cacheLibInit()

**NAME** `cacheLibInit()` – initialize the cache library for a processor architecture

**SYNOPSIS**

```
STATUS cacheLibInit
(
    CACHE_MODE instMode,      /* inst cache mode */
    CACHE_MODE dataMode      /* data cache mode */
)
```

**DESCRIPTION** This routine initializes the function pointers for the appropriate cache library. For architectures with more than one cache implementation, the board support package must select the appropriate cache library with `sysCacheLibInit`. Systems without cache coherency problems (*i.e.*, bus snooping) should NULLify the flush and invalidate function pointers in the `cacheLib` structure to enhance driver and overall system performance. This can be done in `sysHwInit()`.

**RETURNS** OK, or **ERROR** if there is no cache library installed.

**SEE ALSO** `cacheLib`

## cacheLock()

**NAME** cacheLock() – lock all or part of a specified cache

**SYNOPSIS**

```
STATUS cacheLock
(
    CACHE_TYPE cache,          /* cache to lock */
    void *      address,      /* virtual address */
    size_t     bytes         /* number of bytes to lock */
)
```

**DESCRIPTION** This routine locks all (global) or some (local) entries in the specified cache. Cache locking is useful in real-time systems. Not all caches can perform locking.

**RETURNS** OK, or ERROR if the cache type is invalid or the cache control is not supported.

**SEE ALSO** cacheLib

---

## cacheMb930ClearLine()

**NAME** cacheMb930ClearLine() – clear a line from an MB86930 cache

**SYNOPSIS**

```
STATUS cacheMb930ClearLine
(
    CACHE_TYPE cache,          /* cache to clear entry */
    void *      address       /* virtual address */
)
```

**DESCRIPTION** This routine flushes and invalidates a specified line from the specified MB86930 cache.

**RETURNS** OK, or ERROR if the cache type is invalid or the cache control is not supported.

**SEE ALSO** cacheMb930Lib

---

## cacheMb930LibInit()

**NAME** `cacheMb930LibInit()` – initialize the Fujitsu MB86930 cache library

**SYNOPSIS**

```
STATUS cacheMb930LibInit
(
    CACHE_MODE instMode,      /* instruction cache mode */
    CACHE_MODE dataMode      /* data cache mode */
)
```

**DESCRIPTION** This routine installs the function pointers for the Fujitsu MB86930 cache library and performs other necessary cache library initialization. The board support package selects this cache library by setting the function pointer `sysCacheLibInit` equal to `cacheMb930LibInit()`. Note that `sysCacheLibInit` must be initialized on declaration, placing it in the “.data” section.

This routine invalidates the cache tags and leaves the cache disabled. It should only be called during initialization, before any cache locking has taken place.

The only available mode for the MB86930 is `CACHE_WRITETHROUGH`.

**RETURNS** `OK`, or `ERROR` if cache control is not supported.

**SEE ALSO** `cacheMb930Lib`

---

## cacheMb930LockAuto()

**NAME** `cacheMb930LockAuto()` – enable MB86930 automatic locking of kernel instructions/data

**SYNOPSIS**

```
void cacheMb930LockAuto (void)
```

**DESCRIPTION** This routine enables automatic cache locking of kernel instructions and data into MB86930 caches. Once entries are locked into the caches, they cannot be unlocked.

**RETURNS** N/A

**SEE ALSO** `cacheMb930Lib`

## cachePipeFlush()

<b>NAME</b>	cachePipeFlush() – flush processor write buffers to memory
<b>SYNOPSIS</b>	<b>STATUS</b> cachePipeFlush (void)
<b>DESCRIPTION</b>	This routine forces the processor output buffers to write their contents to RAM. A cache flush may have forced its data into the write buffers, then the buffers need to be flushed to RAM to maintain coherency.
<b>RETURNS</b>	OK, or ERROR if the cache control is not supported.
<b>SEE ALSO</b>	cacheLib

---

## cacheR3kLibInit()

<b>NAME</b>	cacheR3kLibInit() – initialize the R3000 cache library
<b>SYNOPSIS</b>	<b>STATUS</b> cacheR3kLibInit ( <b>CACHE_MODE</b> instMode,       /* instruction cache mode */ <b>CACHE_MODE</b> dataMode       /* data cache mode */ )
<b>DESCRIPTION</b>	This routine initializes the function pointers for the R3000 cache library. The board support package can select this cache library by calling this routine.
<b>RETURNS</b>	OK.
<b>SEE ALSO</b>	cacheR3kLib

---

## cacheR4kLibInit()

**NAME** cacheR4kLibInit() – initialize the R4000 cache library

**SYNOPSIS**

```
STATUS cacheR4kLibInit
(
    CACHE_MODE instMode,      /* instruction cache mode */
    CACHE_MODE dataMode,     /* data cache mode */
    UINT32      iCacheSize,
    UINT32      iCacheLineSize,
    UINT32      dCacheSize,
    UINT32      dCacheLineSize,
    UINT32      sCacheSize,
    UINT32      sCacheLineSize
)
```

**DESCRIPTION** This routine initializes the function pointers for the R4000 cache library. The board support package can select this cache library by assigning the function pointer *sysCacheLibInit* to **cacheR4kLibInit()**.

**RETURNS** OK.

**SEE ALSO** cacheR4kLib

---

## cacheR5kLibInit()

**NAME** cacheR5kLibInit() – initialize the R5000 cache library

**SYNOPSIS**

```
STATUS cacheR5kLibInit
(
    CACHE_MODE instMode,      /* instruction cache mode */
    CACHE_MODE dataMode,     /* data cache mode */
    UINT32      iCacheSize,
    UINT32      iCacheLineSize,
    UINT32      dCacheSize,
    UINT32      dCacheLineSize,
    UINT32      sCacheSize,
    UINT32      sCacheLineSize
)
```

**cacheR7kLibInit()**

<b>DESCRIPTION</b>	This routine initializes the function pointers for the R5000 cache library. The board support package can select this cache library by assigning the function pointer <i>sysCacheLibInit</i> to <b>cacheR5kLibInit()</b> .
<b>RETURNS</b>	OK.
<b>SEE ALSO</b>	cacheR5kLib

---

## cacheR7kLibInit()

**NAME** cacheR7kLibInit() – initialize the R7000 cache library

**SYNOPSIS**

```

STATUS cacheR7kLibInit
(
    CACHE_MODE instMode,      /* instruction cache mode */
    CACHE_MODE dataMode,     /* data cache mode */
    UINT32      iCacheSize,
    UINT32      iCacheLineSize,
    UINT32      dCacheSize,
    UINT32      dCacheLineSize,
    UINT32      sCacheSize,
    UINT32      sCacheLineSize,
    UINT32      tCacheSize,
    UINT32      tCacheLineSize
)

```

<b>DESCRIPTION</b>	This routine initializes the function pointers for the R7000 cache library. The board support package can select this cache library by assigning the function pointer <i>sysCacheLibInit</i> to <b>cacheR7kLibInit()</b> .
<b>RETURNS</b>	OK.
<b>SEE ALSO</b>	cacheR7kLib

---

## cacheR10kLibInit()

**NAME** cacheR10kLibInit() – initialize the R10000 cache library

**SYNOPSIS**

```
STATUS cacheR10kLibInit
(
    CACHE_MODE instMode,      /* instruction cache mode */
    CACHE_MODE dataMode,     /* data cache mode */
    UINT32      iCacheSize,
    UINT32      iCacheLineSize,
    UINT32      dCacheSize,
    UINT32      dCacheLineSize,
    UINT32      sCacheSize,
    UINT32      sCacheLineSize
)
```

**DESCRIPTION** This routine initializes the function pointers for the R10000 cache library. The board support package can select this cache library by assigning the function pointer *sysCacheLibInit* to **cacheR10kLibInit()**.

**RETURNS** OK.

**SEE ALSO** cacheR10kLib

---

## cacheR32kLibInit()

**NAME** cacheR32kLibInit() – initialize the RC32364 cache library

**SYNOPSIS**

```
STATUS cacheR32kLibInit
(
    CACHE_MODE instMode,     /* instruction cache mode */
    CACHE_MODE dataMode     /* data cache mode */
)
```

**DESCRIPTION** This routine initializes the function pointers for the RC32364 cache library. The board support package can select this cache library by assigning the function pointer *sysCacheLibInit* to **cacheR32kLibInit()**.

This routine determines the cache size and cache line size for the instruction and data cache automatically by reading the CP0 configuration register. This is different than most

of the other cache library initialization calls, which take the cache and line sizes as parameters.

**RETURNS** OK.

**SEE ALSO** cacheR32kLib

---

## cacheR32kMalloc()

**NAME** cacheR32kMalloc() – allocate a cache-safe buffer, if possible

**SYNOPSIS**

```
void * cacheR32kMalloc
(
    size_t bytes
)
```

**DESCRIPTION** This routine will attempt to return a pointer to a section of memory that will not experience any cache coherency problems.

**RETURNS** A pointer to the non-cached buffer, or NULL.

**SEE ALSO** cacheR32kLib

---

## cacheR33kLibInit()

**NAME** cacheR33kLibInit() – initialize the R33000 cache library

**SYNOPSIS**

```
STATUS cacheR33kLibInit
(
    CACHE_MODE instMode,    /* instruction cache mode */
    CACHE_MODE dataMode    /* data cache mode */
)
```

**DESCRIPTION** This routine initializes the function pointers for the R33000 cache library. The board support package can select this cache library by calling this routine.

**RETURNS** OK.

**SEE ALSO** cacheR33kLib

---

## cacheR333x0LibInit( )

**NAME** cacheR333x0LibInit( ) – initialize the R333x0 cache library

**SYNOPSIS**

```
STATUS cacheR333x0LibInit
(
    CACHE_MODE instMode,    /* instruction cache mode */
    CACHE_MODE dataMode    /* data cache mode */
)
```

**DESCRIPTION** This routine initializes the function pointers for the R333x0 cache library. The board support package can select this cache library by calling this routine.

**RETURNS** OK.

**SEE ALSO** cacheR333x0Lib

---

## cacheSh7040LibInit( )

**NAME** cacheSh7040LibInit( ) – initialize the SH7040 cache library

**SYNOPSIS**

```
STATUS cacheSh7040LibInit
(
    CACHE_MODE instMode,    /* instruction cache mode */
    CACHE_MODE dataMode    /* data cache mode (ignored) */
)
```

**DESCRIPTION** This routine initializes the cache library for the Hitachi SH7040 processors. It initializes the function pointers and configures the caches to the specified cache modes. Modes should be set before caching is enabled. If two complementary flags are set (enable/disable), no action is taken for any of the input flags.

Next caching modes are available for the SH7040 processors:

SH7040:	CACHE_WRITETHROUGH	(cache for instruction)
	CACHE_SH7040_DRAM	(enable caching for DRAM space)
	CACHE_SH7040_CS3	(enable caching for CS3 space)
	CACHE_SH7040_CS2	(enable caching for CS2 space)
	CACHE_SH7040_CS1	(enable caching for CS1 space)
	CACHE_SH7040_CS0	(enable caching for CS0 space)

**RETURNS** OK, or ERROR if the specified caching modes were invalid.

**SEE ALSO** cacheSh7040Lib

---

## cacheSh7604LibInit()

**NAME** cacheSh7604LibInit() – initialize the SH7604/SH7615 cache library

**SYNOPSIS**

```
STATUS cacheSh7604LibInit
(
    CACHE_MODE instMode,      /* instruction cache mode (ignored) */
    CACHE_MODE dataMode      /* data cache mode */
)
```

**DESCRIPTION** This routine initializes the cache library for the Hitachi SH7604/SH7615 processor. It initializes the function pointers and configures the caches to the specified cache modes. Modes should be set before caching is enabled.

The following caching modes are available for the SH7604/SH7615 processor:

SH7604:    **CACHE\_WRITETHROUGH**   (cache for instruction and data)  
          **CACHE\_2WAY\_MODE**       (2KB 2-way cache + 2KB RAM)

**RETURNS** OK, or ERROR if the specified caching modes were invalid.

**SEE ALSO** cacheSh7604Lib

---

## cacheSh7622LibInit()

**NAME** cacheSh7622LibInit() – initialize the SH7622 cache library

**SYNOPSIS**

```
STATUS cacheSh7622LibInit
(
    CACHE_MODE instMode,      /* instruction cache mode */
    CACHE_MODE dataMode      /* data cache mode */
)
```

**DESCRIPTION** This routine initializes the cache library for the Hitachi SH7622 processor. It initializes the function pointers and configures the caches to the specified cache modes. Modes should

be set before caching is enabled. If two complementary flags are set (enable/disable), no action is taken for any of the input flags. Data cache and instruction cache are mixed together in the SH7622.

Next caching modes are available for the SH7622 processor:

SH7622:    **CACHE\_WRITETHROUGH**    (cache for instruction and data)  
           **CACHE\_COPYBACK\_P1**     (write-back cache for P1)

**RETURNS**            OK, or **ERROR** if the specified caching modes were invalid.

**SEE ALSO**            **cacheSh7622Lib**

---

## cacheSh7700LibInit()

**NAME**                **cacheSh7700LibInit()** – initialize the SH7700 cache library

**SYNOPSIS**            **STATUS cacheSh7700LibInit**  
                   (  
                   **CACHE\_MODE instMode,**        /\* instruction cache mode (ignored) \*/  
                   **CACHE\_MODE dataMode**       /\* data cache mode \*/  
                   )

**DESCRIPTION**        This routine initializes the cache library for the Hitachi SH7700 processor. It initializes the function pointers and configures the caches to the specified cache modes. Modes should be set before caching is enabled. If two complementary flags are set (enable/disable), no action is taken for any of the input flags.

The following caching modes are available for the SH7700 processor:

SH7700:    **CACHE\_WRITETHROUGH**    (cache for instruction and data)  
             **CACHE\_COPYBACK**        (cache for instruction and data)  
             **CACHE\_COPYBACK\_P1**    (copy-back cache for P1, SH7709 only)  
             **CACHE\_2WAY\_MODE**        (4KB 2-way cache + 4KB RAM)  
             **CACHE\_1WAY\_MODE**        (2KB direct mapped cache, SH7702 only)  
             **CACHE\_DMA\_BYPASS\_P0**    (allocate DMA buffer to P2, free it to P0)  
             **CACHE\_DMA\_BYPASS\_P1**    (allocate DMA buffer to P2, free it to P1)  
             **CACHE\_DMA\_BYPASS\_P3**    (allocate DMA buffer to P2, free it to P3)

The **CACHE\_DMA\_BYPASS\_Px** modes allow to allocate “cache-safe” buffers without MMU. If none of **CACHE\_DMA\_BYPASS\_Px** modes is specified, **cacheDmaMalloc()** returns a cache-safe buffer on logical space, which is created by the MMU. If **CACHE\_DMA\_BYPASS\_P0** is selected, **cacheDmaMalloc()** returns a cache-safe buffer on

P2 space, and **cacheDmaFree()** releases the buffer to P0 space. Namely, if the system memory partition is located on P0, cache-safe buffers can be allocated and freed without MMU, by selecting **CACHE\_DMA\_BYPASS\_P0**.

**RETURNS** OK, or **ERROR**.

**SEE ALSO** **cacheSh7700Lib**

---

## cacheSh7729LibInit()

**NAME** **cacheSh7729LibInit()** – initialize the SH7729 cache library

**SYNOPSIS**

```
STATUS cacheSh7729LibInit
(
    CACHE_MODE instMode,      /* instruction cache mode (ignored) */
    CACHE_MODE dataMode     /* data cache mode */
)
```

**DESCRIPTION** This routine initializes the cache library for the Hitachi SH7729 processor. It initializes the function pointers and configures the caches to the specified cache modes. Modes should be set before caching is enabled. If two complementary flags are set (enable/disable), no action is taken for any of the input flags.

The following caching modes are available for the SH7729 processor:

SH7729:	<b>CACHE_WRITETHROUGH</b>	(cache for instruction and data)
	<b>CACHE_COPYBACK</b>	(cache for instruction and data)
	<b>CACHE_COPYBACK_P1</b>	(copy-back cache for P1)
	<b>CACHE_DMA_BYPASS_P0</b>	(allocate DMA buffer to P2, free it to P0)
	<b>CACHE_DMA_BYPASS_P1</b>	(allocate DMA buffer to P2, free it to P1)
	<b>CACHE_DMA_BYPASS_P3</b>	(allocate DMA buffer to P2, free it to P3)

The **CACHE\_DMA\_BYPASS\_Px** modes allow to allocate “cache-safe” buffers without MMU. If none of **CACHE\_DMA\_BYPASS\_Px** modes is specified, **cacheDmaMalloc()** returns a cache-safe buffer on logical space, which is created by the MMU. If **CACHE\_DMA\_BYPASS\_P0** is selected, **cacheDmaMalloc()** returns a cache-safe buffer on P2 space, and **cacheDmaFree()** releases the buffer to P0 space. Namely, if the system memory partition is located on P0, cache-safe buffers can be allocated and freed without MMU, by selecting **CACHE\_DMA\_BYPASS\_P0**.

**RETURNS** OK, or **ERROR**.

**SEE ALSO** **cacheSh7729Lib**

---

## cacheSh7750LibInit()

**NAME** cacheSh7750LibInit() – initialize the SH7750 cache library

**SYNOPSIS**

```
STATUS cacheSh7750LibInit
(
    CACHE_MODE instMode,    /* instruction cache mode */
    CACHE_MODE dataMode    /* data cache mode */
)
```

**DESCRIPTION** This routine initializes the cache library for the Hitachi SH7750 processor. It initializes the function pointers and configures the caches to the specified cache modes. Modes should be set before caching is enabled. If two complementary flags are set (enable/disable), no action is taken for any of the input flags.

The following caching modes are available for the SH7750 processor:

SH7750:

<b>CACHE_WRITETHROUGH</b>	
<b>CACHE_COPYBACK</b>	(copy-back cache for P0/P3, data cache only)
<b>CACHE_COPYBACK_P1</b>	(copy-back cache for P1, data cache only)
<b>CACHE_RAM_MODE</b>	(use half of cache as RAM, data cache only)
<b>CACHE_2WAY_MODE</b>	(use RAM in 2way associ. mode, data cache only)
<b>CACHE_A25_INDEX</b>	(use A25 as MSB of cache index)
<b>CACHE_DMA_BYPASS_P0</b>	(allocate DMA buffer to P2, free it to P0)
<b>CACHE_DMA_BYPASS_P1</b>	(allocate DMA buffer to P2, free it to P1)
<b>CACHE_DMA_BYPASS_P3</b>	(allocate DMA buffer to P2, free it to P3)

The **CACHE\_DMA\_BYPASS\_Px** modes allow to allocate “cache-safe” buffers without MMU. If none of **CACHE\_DMA\_BYPASS\_Px** modes is specified, **cacheDmaMalloc()** returns a cache-safe buffer on logical space, which is created by the MMU. If **CACHE\_DMA\_BYPASS\_P0** is selected, **cacheDmaMalloc()** returns a cache-safe buffer on P2 space, and **cacheDmaFree()** releases the buffer to P0 space. Namely, if the system memory partition is located on P0, cache-safe buffers can be allocated and freed without MMU, by selecting **CACHE\_DMA\_BYPASS\_P0**.

**RETURNS** OK, or ERROR if specified cache mode is invalid.

**SEE ALSO** cacheSh7750Lib

---

## cacheStoreBufDisable()

<b>NAME</b>	<code>cacheStoreBufDisable()</code> – disable the store buffer (MC68060 only)
<b>SYNOPSIS</b>	<code>void cacheStoreBufDisable (void)</code>
<b>DESCRIPTION</b>	This routine resets the ESB bit of the Cache Control Register (CACR) to disable the store buffer.
<b>RETURNS</b>	N/A
<b>SEE ALSO</b>	<code>cacheArchLib</code>

---

## cacheStoreBufEnable()

<b>NAME</b>	<code>cacheStoreBufEnable()</code> – enable the store buffer (MC68060 only)
<b>SYNOPSIS</b>	<code>void cacheStoreBufEnable (void)</code>
<b>DESCRIPTION</b>	This routine sets the ESB bit of the Cache Control Register (CACR) to enable the store buffer. To maximize performance, the four-entry first-in-first-out (FIFO) store buffer is used to defer pending writes to writethrough or cache-inhibited imprecise pages.
<b>RETURNS</b>	N/A
<b>SEE ALSO</b>	<code>cacheArchLib</code>

---

## cacheSun4ClearContext()

<b>NAME</b>	<code>cacheSun4ClearContext()</code> – clear a specific context from a Sun-4 cache
<b>SYNOPSIS</b>	<pre>STATUS cacheSun4ClearContext (     CACHE_TYPE cache,      /* cache to clear */     void *      address    /* virtual address */ )</pre>

**DESCRIPTION** This routine flushes and invalidates a specified context from the specified Sun-4 cache.

**RETURNS** OK, or **ERROR** if the cache type is invalid or the cache control is not supported.

**SEE ALSO** cacheSun4Lib

---

## cacheSun4ClearLine()

**NAME** cacheSun4ClearLine() – clear a line from a Sun-4 cache

**SYNOPSIS**

```
STATUS cacheSun4ClearLine
(
    CACHE_TYPE cache,          /* cache to clear */
    void *      address        /* virtual address */
)
```

**DESCRIPTION** This routine flushes and invalidates a specified line from the specified Sun-4 cache.

**RETURNS** OK, or **ERROR** if the cache type is invalid or the cache control is not supported.

**SEE ALSO** cacheSun4Lib

---

## cacheSun4ClearPage()

**NAME** cacheSun4ClearPage() – clear a page from a Sun-4 cache

**SYNOPSIS**

```
STATUS cacheSun4ClearPage
(
    CACHE_TYPE cache,          /* cache to clear */
    void *      address        /* virtual address */
)
```

**DESCRIPTION** This routine flushes and invalidates a specified page from the specified Sun-4 cache.

**RETURNS** OK, or **ERROR** if the cache type is invalid or the cache control is not supported.

**SEE ALSO** cacheSun4Lib

## cacheSun4ClearSegment()

**NAME** cacheSun4ClearSegment() – clear a segment from a Sun-4 cache

**SYNOPSIS**

```
STATUS cacheSun4ClearSegment
(
    CACHE_TYPE cache,      /* cache to clear */
    void *      address    /* virtual address */
)
```

**DESCRIPTION** This routine flushes and invalidates a specified segment from the specified Sun-4 cache.

**RETURNS** OK, or ERROR if the cache type is invalid or the cache control is not supported.

**SEE ALSO** cacheSun4Lib

---

## cacheSun4LibInit()

**NAME** cacheSun4LibInit() – initialize the Sun-4 cache library

**SYNOPSIS**

```
STATUS cacheSun4LibInit
(
    CACHE_MODE instMode,   /* instruction cache mode */
    CACHE_MODE dataMode   /* data cache mode */
)
```

**DESCRIPTION** This routine initializes the function pointers for the Sun Microsystems Sun-4 cache library. The board support package can select this cache library by assigning the function pointer `sysCacheLibInit` to `cacheSun4LibInit()`.

The only available mode for the Sun-4 cache is `CACHE_WRITETHROUGH`.

**RETURNS** OK, or ERROR if cache control is not supported.

**SEE ALSO** cacheSun4Lib

---

## cacheTextUpdate()

**NAME** `cacheTextUpdate()` – synchronize the instruction and data caches

**SYNOPSIS**

```
STATUS cacheTextUpdate
(
    void * address,          /* virtual address */
    size_t bytes            /* number of bytes to sync */
)
```

**DESCRIPTION** This routine flushes the data cache, then invalidates the instruction cache. This operation forces the instruction cache to fetch code that may have been created via the data path.

**RETURNS** OK, or ERROR if the cache control is not supported.

**SEE ALSO** `cacheLib`

---

## cacheTiTms390LibInit()

**NAME** `cacheTiTms390LibInit()` – initialize the TI TMS390 cache library

**SYNOPSIS**

```
STATUS cacheTiTms390LibInit
(
    CACHE_MODE instMode,    /* instruction cache mode */
    CACHE_MODE dataMode    /* data cache mode */
)
```

**DESCRIPTION** This routine initializes the function pointers for the TI TMS390 cache library. The board support package can select this cache library by assigning the function pointer `sysCacheLibInit` to `cacheTiTms390LibInit()`.

The only available cache mode is `CACHE_COPYBACK`.

**RETURNS** OK, or ERROR if cache control is not supported.

**SEE ALSO** `cacheTiTms390Lib`

## **cacheTiTms390PhysToVirt()**

<b>NAME</b>	<b>cacheTiTms390PhysToVirt()</b> – translate a physical address for drivers
<b>SYNOPSIS</b>	<pre>void * cacheTiTms390PhysToVirt (     void * address          /* physical address */ )</pre>
<b>DESCRIPTION</b>	<p>This routine performs a 32-bit physical to 32-bit virtual address translation in the current context.</p> <p>It works for only DRAM addresses of the first EMC.</p> <p>It guesses likely virtual addresses, and checks its guesses with <b>VM_TRANSLATE</b>. A likely virtual address is the same as the physical address, or some multiple of 16M less. If any match, it succeeds. If all guesses are wrong, it fails.</p>
<b>RETURNS</b>	The virtual address that maps to the physical address bits [31:0] argument, or <b>NULL</b> if it fails.
<b>SEE ALSO</b>	<b>cacheTiTms390Lib</b>

---

## **cacheTiTms390VirtToPhys()**

<b>NAME</b>	<b>cacheTiTms390VirtToPhys()</b> – translate a virtual address for <b>cacheLib</b>
<b>SYNOPSIS</b>	<pre>void * cacheTiTms390VirtToPhys (     void * address          /* virtual address */ )</pre>
<b>DESCRIPTION</b>	<p>This routine performs a 32-bit virtual to 32-bit physical address translation in the current context.</p>
<b>RETURNS</b>	The physical address translation bits [31:0] of a virtual address argument, or <b>NULL</b> if the virtual address is not valid, or the physical address does not fit in 32 bits.
<b>RETURNS</b>	N/A
<b>SEE ALSO</b>	<b>cacheTiTms390Lib</b>

---

## cacheTx49LibInit()

**NAME** cacheTx49LibInit() – initialize the Tx49 cache library

**SYNOPSIS**

```
STATUS cacheTx49LibInit
(
    CACHE_MODE instMode,      /* instruction cache mode */
    CACHE_MODE dataMode,     /* data cache mode */
    UINT32      iCacheSize,   /* instruction cache size */
    UINT32      iCacheLineSize, /* instruction cache line size */
    UINT32      dCacheSize,   /* data cache size */
    UINT32      dCacheLineSize /* data cache line size */
)
```

**DESCRIPTION** This routine initializes the function pointers for the Tx49 cache library. The board support package can select this cache library by assigning the function pointer *sysCacheLibInit* to *cacheTx49LibInit()*.

**RETURNS** OK.

**SEE ALSO** cacheTx49Lib

---

## cacheUnlock()

**NAME** cacheUnlock() – unlock all or part of a specified cache

**SYNOPSIS**

```
STATUS cacheUnlock
(
    CACHE_TYPE cache,        /* cache to unlock */
    void *      address,     /* virtual address */
    size_t      bytes       /* number of bytes to unlock */
)
```

**DESCRIPTION** This routine unlocks all (global) or some (local) entries in the specified cache. Not all caches can perform unlocking.

**RETURNS** OK, or **ERROR** if the cache type is invalid or the cache control is not supported.

**SEE ALSO** cacheLib

---

## calloc()

<b>NAME</b>	<b>calloc()</b> – allocate space for an array (ANSI)
<b>SYNOPSIS</b>	<pre>void *calloc (     size_t elemNum,          /* number of elements */     size_t elemSize        /* size of elements */ )</pre>
<b>DESCRIPTION</b>	This routine allocates a block of memory for an array that contains <i>elemNum</i> elements of size <i>elemSize</i> . This space is initialized to zeros.
<b>RETURNS</b>	A pointer to the block, or NULL if the call fails.
<b>SEE ALSO</b>	<b>memLib</b> , <i>American National Standard for Information Systems -Programming Language - C, ANSI X3.159-1989: General Utilities (stdlib.h)</i>

---

## cbioBlkCopy()

<b>NAME</b>	<b>cbioBlkCopy()</b> – block to block (sector to sector) transfer routine
<b>SYNOPSIS</b>	<pre>STATUS cbioBlkCopy (     CBIO_DEV_ID dev,        /* CBIO handle */     block_t   srcBlock,    /* source start block */     block_t   dstBlock,    /* destination start block */     block_t   numBlocks    /* number of blocks to copy */ )</pre>
<b>DESCRIPTION</b>	<p>This routine verifies the CBIO device is valid and if so calls the devices block to block transfer routine which makes copies of one or more blocks on the lower layer (hardware, subordinate CBIO, or <b>BLK_DEV</b>). It is optimized for block to block copies on the subordinate layer.</p> <p>If the <b>CBIO_DEV_ID</b> passed to this routine is not a valid CBIO handle, <b>ERROR</b> will be returned with <b>errno</b> set to <b>S_cbioLib_INVALID_CBIO_DEV_ID</b></p>
<b>RETURNS</b>	<b>OK</b> if successful, or <b>ERROR</b> if the handle is invalid or the CBIO device routine returns <b>ERROR</b> .
<b>SEE ALSO</b>	<b>cbioLib</b>

---

## cbioBlkRW()

**NAME**            `cbioBlkRW()` – transfer blocks to or from memory

**SYNOPSIS**        `STATUS cbioBlkRW`

```
(
    CBIO_DEV_ID dev,          /* CBIO handle */
    block_t     startBlock,  /* starting block of transfer */
    block_t     numBlocks,   /* number of blocks to transfer */
    addr_t      buffer,      /* address of the memory buffer */
    CBIO_RW     rw,          /* direction of transfer R/W */
    cookie_t *  pCookie      /* pointer to cookie */
)
```

**DESCRIPTION**    This routine verifies the CBIO device is valid and if so calls the devices block transfer routine. The CBIO device performs block transfers between the device and memory.

If the `CBIO_DEV_ID` passed to this routine is not a valid CBIO handle, **ERROR** will be returned with **errno** set to `S_cbioLib_INVALID_CBIO_DEV_ID`

**RETURNS**        **OK** if successful or **ERROR** if the handle is invalid, or if the CBIO device routine returns **ERROR**.

**SEE ALSO**        `cbioLib`

---

## cbioBytesRW()

**NAME**            `cbioBytesRW()` – transfer bytes to or from memory

**SYNOPSIS**        `STATUS cbioBytesRW`

```
(
    CBIO_DEV_ID dev,          /* CBIO handle */
    block_t     startBlock,  /* starting block of the transfer */
    off_t       offset,      /* offset into block in bytes */
    addr_t      buffer,      /* address of data buffer */
    size_t      nBytes,      /* number of bytes to transfer */
    CBIO_RW     rw,          /* direction of transfer R/W */
    cookie_t *  pCookie      /* pointer to cookie */
)
```

**cbioDevCreate()**

<b>DESCRIPTION</b>	This routine verifies the CBIO device is valid and if so calls the devices byte transfer routine which transfers between a user buffer and the lower layer (hardware, subordinate CBIO, or <b>BLK_DEV</b> ). It is optimized for byte transfers.  If the <b>CBIO_DEV_ID</b> passed to this routine is not a valid CBIO handle, <b>ERROR</b> will be returned with <b>errno</b> set to <b>S_cbioLib_INVALID_CBIO_DEV_ID</b>
<b>RETURNS</b>	<b>OK</b> if successful or <b>ERROR</b> if the handle is invalid, or if the CBIO device routine returns <b>ERROR</b> .
<b>SEE ALSO</b>	<b>cbioLib</b>

---

## cbioDevCreate()

<b>NAME</b>	<b>cbioDevCreate()</b> – initialize a CBIO device (Generic)
<b>SYNOPSIS</b>	<pre> <b>CBIO_DEV_ID</b> cbioDevCreate     (         <b>caddr_t</b> ramAddr,           /* where it is in memory (0 = KHEAP_ALLOC) */         <b>size_t</b>  ramSize         /* pool size */     ) </pre>
<b>DESCRIPTION</b>	This routine will create an empty <b>CBIO_DEV</b> structure and return a handle to that structure ( <b>CBIO_DEV_ID</b> ).  This routine is intended to be used by CBIO modules only. See <b>cbioLibP.h</b>
<b>RETURNS</b>	<b>CBIO_DEV_ID</b> or <b>NULL</b> if <b>ERROR</b> .
<b>SEE ALSO</b>	<b>cbioLib</b>

---

## cbioDevVerify()

<b>NAME</b>	<b>cbioDevVerify()</b> – verify <b>CBIO_DEV_ID</b>
<b>SYNOPSIS</b>	<pre> <b>STATUS</b> cbioDevVerify     (         <b>CBIO_DEV_ID</b> device       /* CBIO_DEV_ID to be verified */     ) </pre>

- DESCRIPTION** The purpose of this function is to determine if the device complies with the CBIO interface. It can be used to verify a CBIO handle before it is passed to **dosFsLib**, **rawFsLib**, **usrFdiskPartLib**, or other CBIO modules which expect a valid CBIO interface.
- The device handle provided to this function, *device* is verified to be a CBIO device. If *device* is not a CBIO device **ERROR** is returned with **errno** set to **S\_cbioLib\_INVALID\_CBIO\_DEV\_ID**
- The **dcacheCbio** and **dpartCbio** CBIO modules (and **dosFsLib**) use this function internally, and therefore this function need not be otherwise invoked when using compliant CBIO modules.
- RETURNS** **OK** or **ERROR** if not a CBIO device, if passed a **NULL** address, or if the check could cause an unaligned access.
- SEE ALSO** **cbioLib**, **dosFsLib**, **dcacheCbio**, **dpartCbio**

---

## cbioIoctl()

**NAME** **cbioIoctl()** – perform ioctl operation on device

**SYNOPSIS**

```
STATUS cbioIoctl
(
    CBIO_DEV_ID dev,          /* CBIO handle */
    int          command,    /* ioctl command to be issued */
    addr_t      arg         /* arg - specific to ioctl */
)
```

**DESCRIPTION** This routine verifies the CBIO device is valid and if so calls the devices I/O control operation routine.

CBIO modules expect the following **ioctl()** codes:

- **CBIO\_RESET** - reset the CBIO device. When the third argument to the **ioctl** call accompanying **CBIO\_RESET** is **NULL**, the code verifies that the disk is inserted and is ready, after getting it to a known state. When the 3rd argument is a non-zero, it is assumed to be a **BLK\_DEV** pointer and **CBIO\_RESET** will install a new subordinate block device. This work is performed at the **BLK\_DEV** to CBIO layer, and all layers shall account for it. A **CBIO\_RESET** indicates a possible change in device geometry, and the **CBIO\_PARAMS** members will be reinitialized after a **CBIO\_RESET**.
- **CBIO\_STATUS\_CHK** - check device status of CBIO device and lower layer
- **CBIO\_DEVICE\_LOCK** - Prevent disk removal
- **CBIO\_DEVICE\_UNLOCK** - Allow disk removal

## ***cbioLibInit()***

- **CBIO\_DEVICE\_EJECT** - Unmount and eject device
- **CBIO\_CACHE\_FLUSH** - Flush any dirty cached data
- **CBIO\_CACHE\_INVALID** - Flush & Invalidate all cached data
- **CBIO\_CACHE\_NEWBLK** - Allocate scratch block

If the **CBIO\_DEV\_ID** passed to this routine is not a valid CBIO handle, **ERROR** will be returned with **errno** set to **S\_cbioLib\_INVALID\_CBIO\_DEV\_ID**

**RETURNS**            **OK** if successful or **ERROR** if the handle is invalid, or if the CBIO device routine returns **ERROR**.

**SEE ALSO**            **cbioLib**

---

## **cbioLibInit()**

**NAME**                **cbioLibInit()** – Initialize CBIO Library

**SYNOPSIS**            **STATUS cbioLibInit (void)**

**DESCRIPTION**        This function initializes the CBIO library, and will be called when the first CBIO device is created, hence it does not need to be called during system initialization. It can be called multiple times, but will do nothing after the first call.

**RETURNS**            **OK** or **ERROR**

**SEE ALSO**            **cbioLib**

---

## **cbioLock()**

**NAME**                **cbioLock()** – obtain CBIO device semaphore.

**SYNOPSIS**            **STATUS cbioLock**  
                      (  
                      **CBIO\_DEV\_ID dev,**            **/\* CBIO handle \*/**  
                      **int            timeout**            **/\* timeout in ticks \*/**  
                      )

<b>DESCRIPTION</b>	If the <code>CBIO_DEV_ID</code> passed to this routine is not a valid CBIO handle, <b>ERROR</b> will be returned with <b>errno</b> set to <code>S_cbioLib_INVALID_CBIO_DEV_ID</code>
<b>RETURNS</b>	OK or <b>ERROR</b> if the CBIO handle is invalid or <code>semTake()</code> fails.
<b>SEE ALSO</b>	<code>cbioLib</code>

---

## **cbioModeGet()**

**NAME** `cbioModeGet()` – return the mode setting for CBIO device

**SYNOPSIS**

```
int cbioModeGet
(
    CBIO_DEV_ID dev          /* CBIO handle */
)
```

**DESCRIPTION** If the `CBIO_DEV_ID` passed to this routine is not a valid CBIO handle, **ERROR** will be returned with **errno** set to `S_cbioLib_INVALID_CBIO_DEV_ID`. This routine is not protected by a semaphore.

This routine confirms if the current layer is a CBIO to BLKDEV wrapper or a CBIO to CBIO layer. Depending on the current layer it either returns the mode from `BLK_DEV` or calls `cbioModeGet()` recursively.

**RETURNS** `O_RDONLY`, `O_WRONLY`, or `O_RDWR` or **ERROR**

**SEE ALSO** `cbioLib`

---

## **cbioModeSet()**

**NAME** `cbioModeSet()` – set mode for CBIO device

**SYNOPSIS**

```
STATUS cbioModeSet
(
    CBIO_DEV_ID dev,          /* CBIO handle */
    int mode                  /* O_RDONLY, O_WRONLY, or O_RDWR */
)
```

**DESCRIPTION** Valid modes are `O_RDONLY`, `O_WRONLY`, or `O_RDWR`.

If the `CBIO_DEV_ID` passed to this routine is not a valid CBIO handle, **ERROR** will be returned with `errno` set to `S_cbioLib_INVALID_CBIO_DEV_ID`. This routine is not protected by a semaphore.

This routine confirms if the current layer is a CBIO to BLKDEV wrapper or a CBIO to CBIO layer. Depending on the current layer it either sets the mode of the `BLK_DEV` or calls `cbioModeSet()` recursively.

**RETURNS** OK or **ERROR** if mode is not set.

**SEE ALSO** `cbioLib`

---

## **cbioParamsGet()**

**NAME** `cbioParamsGet()` – fill in `CBIO_PARAMS` structure with CBIO device parameters

**SYNOPSIS**

```
STATUS cbioParamsGet
(
    CBIO_DEV_ID dev,          /* CBIO handle */
    CBIO_PARAMS * pCbioParams /* pointer to CBIO_PARAMS */
)
```

**DESCRIPTION** If the `CBIO_DEV_ID` passed to this routine is not a valid CBIO handle, **ERROR** will be returned with `errno` set to `S_cbioLib_INVALID_CBIO_DEV_ID`.

**RETURNS** OK or **ERROR** if the CBIO handle is invalid.

**SEE ALSO** `cbioLib`

---

## **cbioRdyChgdGet()**

**NAME** `cbioRdyChgdGet()` – determine ready status of CBIO device

**SYNOPSIS**

```
int cbioRdyChgdGet
(
    CBIO_DEV_ID dev          /* CBIO handle */
)
```

**DESCRIPTION** For example

```

switch (cbioRdyChgdGet (cbioDeviceId))
{
case TRUE:
    printf ("Disk changed.\n");
    break;
case FALSE:
    printf ("Disk has not changed.\n");
    break;
case ERROR:
    printf ("Not a valid CBIO device.\n");
    break;
default:
    break;
}

```

If the `CBIO_DEV_ID` passed to this routine is not a valid CBIO handle, `ERROR` will be returned with `errno` set to `S_cbioLib_INVALID_CBIO_DEV_ID`. This routine is not protected by a semaphore.

This routine will check down to the driver layer to see if any lower layer has its ready changed bit set to `TRUE`. If so, this routine returns `TRUE`. If no lower layer has its ready changed bit set to `TRUE`, this layer returns `FALSE`.

**RETURNS** `TRUE` if device ready status has changed, else `FALSE` if the ready status has not changed, else `ERROR` if the `CBIO_DEV_ID` is invalid.

**SEE ALSO** `cbioLib`

---

## cbioRdyChgdSet()

**NAME** `cbioRdyChgdSet()` – force a change in ready status of CBIO device

**SYNOPSIS**

```

STATUS cbioRdyChgdSet
(
    CBIO_DEV_ID dev,          /* CBIO handle */
    BOOL          status     /* TRUE/FALSE */
)

```

**DESCRIPTION** Pass `TRUE` in status to force `READY` status change.

If the `CBIO_DEV_ID` passed to this routine is not a valid CBIO handle, `ERROR` will be returned with `errno` set to `S_cbioLib_INVALID_CBIO_DEV_ID`. If status is not passed as `TRUE` or `FALSE`, `ERROR` is returned. This routine is not protected by a semaphore.

This routine sets *readyChanged* bit of passed `CBIO_DEV`.

**RETURNS** OK or **ERROR** if the device is invalid or status is not **TRUE** or **FALSE**.

**SEE ALSO** **cbioLib**

---

## **cbioShow()**

**NAME** **cbioShow()** – print information about a CBIO device

**SYNOPSIS**

```
STATUS cbioShow  
(  
    CBIO_DEV_ID dev           /* CBIO handle */  
)
```

**DESCRIPTION** This function will display on standard output all information which is generic for all CBIO devices. See the CBIO modules particular device show routines for displaying implementation-specific information.

It takes two arguments:

A **CBIO\_DEV\_ID** which is the CBIO handle to display or **NULL** for the most recent device.

**RETURNS** OK or **ERROR** if no valid **CBIO\_DEV** is found.

**SEE ALSO** **cbioLib**, **dcacheShow()**, **dpartShow()**

---

## **cbioUnlock()**

**NAME** **cbioUnlock()** – release CBIO device semaphore.

**SYNOPSIS**

```
STATUS cbioUnlock  
(  
    CBIO_DEV_ID dev           /* CBIO handle */  
)
```

**DESCRIPTION** If the **CBIO\_DEV\_ID** passed to this routine is not a valid CBIO handle, **ERROR** will be returned with **errno** set to **S\_cbioLib\_INVALID\_CBIO\_DEV\_ID**

**RETURNS** OK or **ERROR** if the CBIO handle is invalid or the **semGive()** fails.

**SEE ALSO** **cbioLib**

---

## cbioWrapBlkDev()

**NAME**            **cbioWrapBlkDev()** – create CPIO wrapper atop a **BLK\_DEV** device

**SYNOPSIS**        **CBIO\_DEV\_ID cbioWrapBlkDev**  
                  (  
                  **BLK\_DEV \* pDevice**            */\* BLK\_DEV \* device pointer \*/*  
                  )

**DESCRIPTION**    The purpose of this function is to make a blkIo (**BLK\_DEV**) device comply with the CPIO interface via a wrapper.

The device handle provided to this function, *device* is verified to be a blkIo device. A lean CPIO to **BLK\_DEV** wrapper is then created for a valid blkIo device. The returned **CBIO\_DEV\_ID** device handle may be used with **dosFsDevCreate()**, **dcacheDevCreate()**, and any other routine expecting a valid **CBIO\_DEV\_ID** handle.

To verify a blkIo pointer we see that all mandatory functions are not **NULL**.

Note that if a valid **CBIO\_DEV\_ID** is passed to this function, it will simply be returned without modification.

The **dosFsLib**, **dcacheCbio**, and **dpartCbio** CPIO modules use this function internally, and therefore this function need not be otherwise invoked when using those CPIO modules.

**RETURNS**        a CPIO device pointer, or **NULL** if not a blkIo device

**SEE ALSO**        **cbioLib**, **dosFsLib**, **dcacheCbio**, **dpartCbio**

---

## cbrt()

**NAME**            **cbrt()** – compute a cube root

**SYNOPSIS**        **double cbrt**  
                  (  
                  **double x**    */\* value to compute the cube root of \*/*  
                  )

**DESCRIPTION**    This routine returns the cube root of *x* in double precision.

**INCLUDE FILES**   **math.h**

**cbrtf()**

**RETURNS** The double-precision cube root of  $x$ .

**SEE ALSO** **mathALib**

**cbrtf()**

**NAME** **cbrtf()** – compute a cube root

**SYNOPSIS**

```
float cbrtf
(
    float x /* argument */
)
```

**DESCRIPTION** This routine returns the cube root of  $x$  in single precision.

**INCLUDE FILES** **math.h**

**RETURNS** The single-precision cube root of  $x$ .

**SEE ALSO** **mathALib**

**cd()**

**NAME** **cd()** – change the default directory

**SYNOPSIS**

```
STATUS cd
(
    const char * name /* new directory name */
)
```

**DESCRIPTION** **NOTE:** This is a target resident function, which manipulates the target I/O system. It must be preceded with the @ letter if executed from the Tornado Shell (**windsh**), which has a built-in command of the same name that operates on the Host's I/O system.

This command sets the default directory to *name*. The default directory is a device name, optionally followed by a directory local to that device.

To change to a different directory, specify one of the following:

- an entire path name with a device name, possibly followed by a directory name. The

entire path name will be changed.

- a directory name starting with a `~` or `/` or `$`. The directory part of the path, immediately after the device name, will be replaced with the new directory name.
- a directory name to be appended to the current default directory. The directory name will be appended to the current default directory.

An instance of `..` indicates one level up in the directory tree.

Note that when accessing a remote file system via RSH or FTP, the VxWorks network device must already have been created using `netDevCreate()`.

---

**WARNING:** The `cd()` command does very little checking that *name* represents a valid path. If the path is invalid, `cd()` may return `OK`, but subsequent calls that depend on the default path will fail.

---

**EXAMPLES**

The following example changes the directory to device `/fd0/`:

```
-> cd "/fd0/"
```

This example changes the directory to device `wrs:` with the local directory `~leslie/target`:

```
-> cd "wrs:~leslie/target"
```

After the previous command, the following changes the directory to `wrs:~leslie/target/config`:

```
-> cd "config"
```

After the previous command, the following changes the directory to `wrs:~leslie/target/demo`:

```
-> cd "../demo"
```

After the previous command, the following changes the directory to `wrs:/etc`.

```
-> cd "/etc"
```

Note that `~` can be used only on network devices (RSH or FTP).

**RETURNS**

`OK` or `ERROR`.

**SEE ALSO**

`usrFsLib`, `pwd()`, *VxWorks Programmer's Guide: Target Shell*

---

## cdromFsDevCreate()

**NAME** `cdromFsDevCreate()` – create a `cdromFsLib` device

**SYNOPSIS**

```
CDROM_VOL_DESC_ID cdromFsDevCreate
(
    char *    devName,        /* device name */
    BLK_DEV * pBlkDev        /* ptr to block device */
)
```

**DESCRIPTION** This routine creates an instance of a `cdromFsLib` device in the I/O system. As input, this function requires a pointer to a `BLK_DEV` structure for the CD-ROM drive on which you want to create a `cdromFsLib` device. Thus, you should already have called `scsiBlkDevCreate()` prior to calling `cdromFsDevCreate()`.

**RETURNS** `CDROM_VOL_DESC_ID`, or `NULL` if error.

**SEE ALSO** `cdromFsLib`, `cdromFsInit()`

---

## cdromFsInit()

**NAME** `cdromFsInit()` – initialize `cdromFsLib`

**SYNOPSIS** `STATUS cdromFsInit (void)`

**DESCRIPTION** This routine initializes `cdromFsLib`. It must be called exactly once before calling any other routine in `cdromFsLib`.

**ERRNO** `S_cdromFsLib_ALREADY_INIT`

**RETURNS** `OK` or `ERROR`, if `cdromFsLib` has already been initialized.

**SEE ALSO** `cdromFsLib`, `cdromFsDevCreate()`, `iosLib.h`

---

## cdromFsVolConfigShow()

<b>NAME</b>	<code>cdromFsVolConfigShow()</code> – show the volume configuration information
<b>SYNOPSIS</b>	<pre>VOID cdromFsVolConfigShow (     void * arg                /* device name or CDROM_VOL_DESC * */ )</pre>
<b>DESCRIPTION</b>	This routine retrieves the volume configuration for the named <code>cdromFsLib</code> device and prints it to standard output. The information displayed is retrieved from the <code>BLK_DEV</code> structure for the specified device.
<b>RETURNS</b>	N/A
<b>SEE ALSO</b>	<code>cdromFsLib</code>

---

## ceil()

<b>NAME</b>	<code>ceil()</code> – compute the smallest integer greater than or equal to a specified value (ANSI)
<b>SYNOPSIS</b>	<pre>double ceil (     double v                  /* value to find the ceiling of */ )</pre>
<b>DESCRIPTION</b>	This routine returns the smallest integer greater than or equal to <i>v</i> , in double precision.
<b>INCLUDE FILES</b>	<code>math.h</code>
<b>RETURNS</b>	The smallest integral value greater than or equal to <i>v</i> , in double precision.
<b>SEE ALSO</b>	<code>ansiMath</code> , <code>mathALib</code>

**ceilf()**

---

**ceilf()**

<b>NAME</b>	<b>ceilf()</b> – compute the smallest integer greater than or equal to a specified value (ANSI)
<b>SYNOPSIS</b>	<pre>float ceilf (     float v    /* value to find the ceiling of */ )</pre>
<b>DESCRIPTION</b>	This routine returns the smallest integer greater than or equal to <i>v</i> , in single precision.
<b>INCLUDE FILES</b>	<b>math.h</b>
<b>RETURNS</b>	The smallest integral value greater than or equal to <i>v</i> , in single precision.
<b>SEE ALSO</b>	<b>mathALib</b>

---

**cfree()**

<b>NAME</b>	<b>cfree()</b> – free a block of memory
<b>SYNOPSIS</b>	<pre>STATUS cfree (     char * pBlock    /* pointer to block of memory to free */ )</pre>
<b>DESCRIPTION</b>	<p>This routine returns to the free memory pool a block of memory previously allocated with <b>calloc()</b>.</p> <p>It is an error to free a memory block that was not previously allocated.</p>
<b>RETURNS</b>	OK, or ERROR if the the block is invalid.
<b>SEE ALSO</b>	<b>memLib</b>

---

## chdir()

**NAME** `chdir()` – set the current default path

**SYNOPSIS**

```
STATUS chdir
(
    char * pathname          /* name of the new default path */
)
```

**DESCRIPTION** This routine sets the default I/O path. All relative pathnames specified to the I/O system will be prepended with this pathname. This pathname must be an absolute pathname, *i.e.*, *name* must begin with an existing device name.

**RETURNS** OK, or ERROR if the first component of the pathname is not an existing device.

**SEE ALSO** `ioLib`, `ioDefPathSet()`, `ioDefPathGet()`, `getcwd()`

---

## checkStack()

**NAME** `checkStack()` – print a summary of each task's stack usage

**SYNOPSIS**

```
void checkStack
(
    int taskNameOrId        /* task name or task ID; 0 = summarize all */
)
```

**DESCRIPTION** This command displays a summary of stack usage for a specified task, or for all tasks if no argument is given. The summary includes the total stack size (SIZE), the current number of stack bytes used (CUR), the maximum number of stack bytes used (HIGH), and the number of bytes never used at the top of the stack (MARGIN = SIZE - HIGH). For example:

```
-> checkStack tShell
      NAME          ENTRY          TID      SIZE    CUR  HIGH  MARGIN
-----
tShell  _shell      23e1c78  9208   832  3632  5576
```

The maximum stack usage is determined by scanning down from the top of the stack for the first byte whose value is not 0xee. In VxWorks, when a task is spawned, all bytes of a task's stack are initialized to 0xee.

**chkdsk()**

<b>DEFICIENCIES</b>	It is possible for a task to write beyond the end of its stack, but not write into the last part of its stack. This will not be detected by <b>checkStack()</b> .
<b>RETURNS</b>	N/A
<b>SEE ALSO</b>	<b>usrLib</b> , <b>taskSpawn()</b> , <i>VxWorks Programmer's Guide: Target Shell</i> , <b>windsh</b> , <i>Tornado User's Guide: Shell</i>

---

**chkdsk()**

**NAME** **chkdsk()** – perform consistency checking on a MS-DOS file system

**SYNOPSIS**

```

STATUS chkdsk
(
    const char * pDevName,    /* device name */
    u_int      repairLevel, /* how to fix errors */
    u_int      verbose      /* verbosity level */
)

```

**DESCRIPTION** This function invokes the integral consistency checking built into the **dosFsLib** file system, via **FIOCHKDSK** ioctl. During the test, the file system will be blocked from application code access, and will emit messages describing any inconsistencies found on the disk, as well as some statistics, depending on the value of the *verbose* argument. Depending the value of *repairLevel*, the inconsistencies will be repaired, and changes written to disk.

These are the values for *repairLevel*:

0	Same as <b>DOS_CHK_ONLY</b> (1)
<b>DOS_CHK_ONLY</b> (1)	Only report errors, do not modify disk.
<b>DOS_CHK_REPAIR</b> (2)	Repair any errors found.

These are the values for *verbose*:

0	similar to <b>DOS_CHK_VERB_1</b>
<b>DOS_CHK_VERB_SILENT</b> (0xff00)	Do not emit any messages, except errors encountered.
<b>DOS_CHK_VERB_1</b> (0x0100)	Display some volume statistics when done testing, as well as errors encountered during the test.
<b>DOS_CHK_VERB_2</b> (0x0200)	In addition to the above option, display path of every file, while it is being checked. This option may significantly slow down the test process.

Note that the consistency check procedure will *unmount* the file system, meaning the all currently open file descriptors will be deemed unusable.

**RETURNS** OK or ERROR if device can not be checked or could not be repaired.

**SEE ALSO** `usrFsLib`

---

## **cleanUpStoreBuffer()**

**NAME** `cleanUpStoreBuffer()` – clean up store buffer after a data store error interrupt

**SYNOPSIS**

```
void cleanUpStoreBuffer
(
    UINT mcntl,           /* Value of MMU Control Register */
    BOOL exception       /* TRUE if exception, FALSE if int */
)
```

**DESCRIPTION** This routine cleans up the store buffer after a data store error interrupt. The first queued store is retried. It is logged as either a recoverable or unrecoverable error. Then the store buffer is re-enabled and other queued stores are processed by the store buffer.

**RETURNS** N/A

**SEE ALSO** `cacheTiTms390Lib`

---

## **clearerr()**

**NAME** `clearerr()` – clear end-of-file and error flags for a stream (ANSI)

**SYNOPSIS**

```
void clearerr
(
    FILE * fp           /* stream to clear EOF and ERROR flags for */
)
```

**DESCRIPTION** This routine clears the end-of-file and error flags for a specified stream.

**INCLUDE FILES** `stdio.h`

**RETURNS** N/A

**SEE ALSO** `ansiStdio`, `feof()`, `ferror()`

**clock()**

---

**clock()****NAME** `clock()` – determine the processor time in use (ANSI)**SYNOPSIS** `clock_t clock (void)`**DESCRIPTION** This routine returns the implementation's best approximation of the processor time used by the program since the beginning of an implementation-defined era related only to the program invocation. To determine the time in seconds, the value returned by `clock()` should be divided by the value of the macro `CLOCKS_PER_SEC`. If the processor time used is not available or its value cannot be represented, `clock()` returns -1.

---

**NOTE:** This routine always returns -1 in VxWorks. VxWorks does not track per-task time or system idle time. There is no method of determining how long a task or the entire system has been doing work. `tickGet()` can be used to query the number of system ticks since system start. `clock_gettime()` can be used to get the current clock time.

---

**INCLUDE FILES** `time.h`**RETURNS** -1**SEE ALSO** `ansiTime`, `tickGet()`, `clock_gettime()`

---

**clock\_getres()****NAME** `clock_getres()` – get the clock resolution (POSIX)**SYNOPSIS**

```
int clock_getres
(
    clockid_t      clock_id, /* clock ID (always CLOCK_REALTIME) */
    struct timespec * res    /* where to store resolution */
)
```

**DESCRIPTION** This routine gets the clock resolution, in nanoseconds, based on the rate returned by `sysClkRateGet()`. If `res` is non-NULL, the resolution is stored in the location pointed to.**RETURNS** 0 (OK), or -1 (ERROR) if `clock_id` is invalid.**ERRNO** EINVAL**SEE ALSO** `clockLib`, `clock_settime()`, `sysClkRateGet()`, `clock_setres()`

---

## clock\_gettime()

**NAME** `clock_gettime()` – get the current time of the clock (POSIX)

**SYNOPSIS**

```
int clock_gettime
(
    clockid_t      clock_id, /* clock ID (always CLOCK_REALTIME) */
    struct timespec * tp      /* where to store current time */
)
```

**DESCRIPTION** This routine gets the current value *tp* for the clock.

**RETURNS** 0 (OK), or -1 (ERROR) if *clock\_id* is invalid or *tp* is NULL.

**ERRNO** EINVAL, EFAULT

**SEE ALSO** `clockLib`

---

## clock\_setres()

**NAME** `clock_setres()` – set the clock resolution

**SYNOPSIS**

```
int clock_setres
(
    clockid_t      clock_id, /* clock ID (always CLOCK_REALTIME) */
    struct timespec * res     /* resolution to be set */
)
```

**DESCRIPTION** This routine is obsolete. It will always return OK.

---

**NOTE:** Non-POSIX.

---

**RETURNS** OK always.

**ERRNO** EINVAL

**SEE ALSO** `clockLib`, `clock_getres()`, `sysClkRateSet()`

## clock\_settime()

<b>NAME</b>	<code>clock_settime()</code> – set the clock to a specified time (POSIX)
<b>SYNOPSIS</b>	<pre>int clock_settime (     clockid_t          clock_id, /* clock ID (always CLOCK_REALTIME) */     const struct timespec * tp    /* time to set */ )</pre>
<b>DESCRIPTION</b>	This routine sets the clock to the value <i>tp</i> , which should be a multiple of the clock resolution. If <i>tp</i> is not a multiple of the resolution, it is truncated to the next smallest multiple of the resolution.
<b>RETURNS</b>	0 (OK), or -1 (ERROR) if <i>clock_id</i> is invalid, <i>tp</i> is outside the supported range, or the <i>tp</i> nanosecond value is less than 0 or equal to or greater than 1,000,000,000.
<b>ERRNO</b>	EINVAL
<b>SEE ALSO</b>	clockLib, clock_getres()

---

## close()

<b>NAME</b>	<code>close()</code> – close a file
<b>SYNOPSIS</b>	<pre>STATUS close (     int fd                /* file descriptor to close */ )</pre>
<b>DESCRIPTION</b>	This routine closes the specified file and frees the file descriptor. It calls the device driver to do the work.
<b>RETURNS</b>	The status of the driver close routine, or ERROR if the file descriptor is invalid.
<b>SEE ALSO</b>	ioLib

---

## closedir()

**NAME** `closedir()` – close a directory (POSIX)

**SYNOPSIS**

```
STATUS closedir
(
    DIR * pDir          /* pointer to directory descriptor */
)
```

**DESCRIPTION** This routine closes a directory which was previously opened using `opendir()`. The `pDir` parameter is the directory descriptor pointer that was returned by `opendir()`.

**RETURNS** OK or ERROR.

**SEE ALSO** `dirLib`, `opendir()`, `readdir()`, `rewinddir()`

---

## connect()

**NAME** `connect()` – initiate a connection to a socket

**SYNOPSIS**

```
STATUS connect
(
    int          s,          /* socket descriptor */
    struct sockaddr * name, /* addr of socket to connect */
    int          namelen /* length of name, in bytes */
)
```

**DESCRIPTION** If `s` is a socket of type `SOCK_STREAM`, this routine establishes a virtual circuit between `s` and another socket specified by `name`. If `s` is of type `SOCK_DGRAM`, it permanently specifies the peer to which messages are sent. If `s` is of type `SOCK_RAW`, it specifies the raw socket upon which data is to be sent and received. The `name` parameter specifies the address of the other socket.

**NOTE:** If a socket with type `SOCK_STREAM` is marked non-blocking, this routine will return **ERROR** with an error number of `EINPROGRESS` or `EALREADY` if a connection attempt is pending. A later call will return **ERROR** and set the error number to `EISCONN` once the connection is established. The connection attempt must be repeated until that result occurs or until this routine establishes a connection immediately and returns **OK**.

**RETURNS** OK, or **ERROR** if the connection attempt does not complete.

**SEE ALSO** `sockLib`

---

## connectWithTimeout()

**NAME** connectWithTimeout() – attempt socket connection within a specified duration

**SYNOPSIS**

```
STATUS connectWithTimeout
(
    int          sock,      /* socket descriptor */
    struct sockaddr * adrs, /* addr of the socket to connect */
    int          adrsLen,  /* length of the socket, in bytes */
    struct timeval * timeVal /* time-out value */
)
```

**DESCRIPTION**

Use this routine as an alternative to **connect()** when your application requires a shorter time out on a connection attempt. By design, a TCP connection attempt times out after 75 seconds if unsuccessful. Thus, a blocking TCP socket **connect()** call might not return for 75 seconds. A **connectWithTimeout()** call lets you reduce this time out by scheduling an abort of the connection attempt if it is not successful before *timeVal*. However, **connectWithTimeout()** does not actually change the TCP timeout value. Thus, you cannot use **connectWithTimeout()** to lengthen the connection time out beyond the TCP default.

In all respects other than the time out value, a **connectWithTimeout()** call behaves exactly like **connect()**. Thus, if no application is listening for connections at the other end, **connectWithTimeout()** returns immediately just like **connect()**. If you specify a NULL pointer for *timeVal*, **connectWithTimeout()** behaves exactly like a **connect()** call.

**RETURNS** OK, or ERROR if a new connection is not established before timeout.

**SEE ALSO** sockLib, connect()

---

## copy()

**NAME** copy() – copy *in* (or stdin) to *out* (or stdout)

**SYNOPSIS**

```
STATUS copy
(
    const char * in,      /* name of file to read (if NULL assume stdin) */
    const char * out     /* name of file to write (if NULL assume */
                        /* stdout) */
)
```

- DESCRIPTION** This command copies from the input file to the output file, until an end-of-file is reached.
- EXAMPLES** The following example displays the file **dog**, found on the default file device:
- ```
-> copy <dog
```
- This example copies from the console to the file **dog**, on device **/ct0/**, until an EOF (default CTRL+D) is typed:
- ```
-> copy >/ct0/dog
```
- This example copies the file **dog**, found on the default file device, to device **/ct0/**:
- ```
-> copy <dog >/ct0/dog
```
- This example makes a conventional copy from the file named **file1** to the file named **file2**:
- ```
-> copy "file1", "file2"
```
- Remember that standard input and output are global; therefore, spawning the first three constructs will not work as expected.
- RETURNS** OK, or **ERROR** if *in* or *out* cannot be opened/created, or if there is an error copying from *in* to *out*.
- SEE ALSO** **usrFsLib**, **copyStreams()**, **tyEOFSet()**, **cp()**, **xcopy()**, *VxWorks Programmer's Guide: Target Shell*

---

## copyStreams()

- NAME** **copyStreams()** – copy from/to specified streams
- SYNOPSIS**
- ```
STATUS copyStreams
(
    int inFd,           /* file descriptor of stream to copy from */
    int outFd          /* file descriptor of stream to copy to */
)
```
- DESCRIPTION** This command copies from the stream identified by *inFd* to the stream identified by *outFd* until an end of file is reached in *inFd*. This command is used by **copy()**.
- RETURNS** OK, or **ERROR** if there is an error reading from *inFd* or writing to *outFd*.
- SEE ALSO** **usrFsLib**, **copy()**, *VxWorks Programmer's Guide: Target Shell*

**cos()**

---

|                      |                                                                                                     |
|----------------------|-----------------------------------------------------------------------------------------------------|
| <b>NAME</b>          | <b>cos()</b> – compute a cosine (ANSI)                                                              |
| <b>SYNOPSIS</b>      | <pre>double cos (     double x          /* angle in radians */ )</pre>                              |
| <b>DESCRIPTION</b>   | This routine computes the cosine of $x$ in double precision. The angle $x$ is expressed in radians. |
| <b>INCLUDE FILES</b> | <b>math.h</b>                                                                                       |
| <b>RETURNS</b>       | The double-precision cosine of $x$ .                                                                |
| <b>SEE ALSO</b>      | <b>ansiMath, mathALib</b>                                                                           |

---

**cosf()**

|                      |                                                                                                    |
|----------------------|----------------------------------------------------------------------------------------------------|
| <b>NAME</b>          | <b>cosf()</b> – compute a cosine (ANSI)                                                            |
| <b>SYNOPSIS</b>      | <pre>float cosf (     float x   /* angle in radians */ )</pre>                                     |
| <b>DESCRIPTION</b>   | This routine returns the cosine of $x$ in single precision. The angle $x$ is expressed in radians. |
| <b>INCLUDE FILES</b> | <b>math.h</b>                                                                                      |
| <b>RETURNS</b>       | The single-precision cosine of $x$ .                                                               |
| <b>SEE ALSO</b>      | <b>mathALib</b>                                                                                    |

---

## cosh()

|                      |                                                                                                                                         |
|----------------------|-----------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>          | <b>cosh()</b> – compute a hyperbolic cosine (ANSI)                                                                                      |
| <b>SYNOPSIS</b>      | <pre>double cosh (     double x          /* value to compute the hyperbolic cosine of */ )</pre>                                        |
| <b>DESCRIPTION</b>   | This routine returns the hyperbolic cosine of $x$ in double precision (IEEE double, 53 bits). A range error occurs if $x$ is too large. |
| <b>INCLUDE FILES</b> | math.h                                                                                                                                  |
| <b>RETURNS</b>       | The double-precision hyperbolic cosine of $x$ .<br>Special cases:<br>If $x$ is +INF, -INF, or NaN, <b>cosh()</b> returns $x$ .          |
| <b>SEE ALSO</b>      | ansiMath, mathALib                                                                                                                      |

---

## coshf()

|                      |                                                                                                                                                                                                               |
|----------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>          | <b>coshf()</b> – compute a hyperbolic cosine (ANSI)                                                                                                                                                           |
| <b>SYNOPSIS</b>      | <pre>float coshf (     float x  /* value to compute the hyperbolic cosine of */ )</pre>                                                                                                                       |
| <b>DESCRIPTION</b>   | This routine returns the hyperbolic cosine of $x$ in single precision.                                                                                                                                        |
| <b>INCLUDE FILES</b> | math.h                                                                                                                                                                                                        |
| <b>RETURNS</b>       | The single-precision hyperbolic cosine of $x$ if the parameter is greater than 1.0, or NaN if the parameter is less than 1.0.<br>Special cases:<br>If $x$ is +INF, -INF, or NaN, <b>coshf()</b> returns $x$ . |
| <b>SEE ALSO</b>      | mathALib                                                                                                                                                                                                      |

**cp()**

---

**cp()**

|                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
|--------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>        | <b>cp()</b> – copy file into other file/directory.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
| <b>SYNOPSIS</b>    | <pre> <b>STATUS</b> cp     (         const char * src,          /* source file or wildcard pattern */         const char * dest         /* destination file name or directory */     ) </pre>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| <b>DESCRIPTION</b> | <p>This command copies from the input file to the output file. If destination name is directory, a source file is copied into this directory, using the last element of the source file name to be the name of the destination file.</p> <p>This function is very similar to <b>copy()</b>, except it is somewhat more similar to the UNIX “cp” program in its handling of the destination.</p> <p><i>src</i> may contain a wildcard pattern, in which case all files matching the pattern will be copied to the directory specified in <i>dest</i>. This function does not copy directories, and is not recursive. To copy entire subdirectories recursively, use <b>xcopy()</b>.</p> |
| <b>EXAMPLES</b>    | <pre> -&gt; cp( "/sd0/FILE1.DAT", "/sd0/dir2/f001.dat") -&gt; cp( "/sd0/dir1/file88", "/sd0/dir2") -&gt; cp( "/sd0/*.tmp", "/sd0/junkdir") </pre>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| <b>RETURNS</b>     | OK or <b>ERROR</b> if destination is not a directory while <i>src</i> is a wildcard pattern, or if any of the files could not be copied.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
| <b>SEE ALSO</b>    | <b>xcopy()</b>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| <b>SEE ALSO</b>    | <b>usrFsLib</b>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |

---

**cplusCallNewHandler()**

|                    |                                                                                                                                                                                                                                                 |
|--------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>        | <b>cplusCallNewHandler()</b> – call the allocation failure handler (C++)                                                                                                                                                                        |
| <b>SYNOPSIS</b>    | <pre> extern void cplusCallNewHandler () </pre>                                                                                                                                                                                                 |
| <b>DESCRIPTION</b> | <p>This function provides a procedural-interface to the new-handler. It can be used by user-defined new operators to call the current new-handler. This function is specific to VxWorks and may not be available in other C++ environments.</p> |

**RETURNS** N/A

**SEE ALSO** **cplusLib**

---

## cplusCtors()

**NAME** **cplusCtors()** – call static constructors (C++)

**SYNOPSIS**

```
extern "C" void cplusCtors
(
    const char * moduleName /* name of loaded module */
)
```

**DESCRIPTION** This function is used to call static constructors under the manual strategy (see **cplusXtorSet()**). *moduleName* is the name of an object module that was “munched” before loading. If *moduleName* is 0, then all static constructors, in all modules loaded by the VxWorks module loader, are called.

**EXAMPLES** The following example shows how to initialize the static objects in modules called “aplx.out” and “apply.out”.

```
-> cplusCtors "aplx.out"
value = 0 = 0x0
-> cplusCtors "apply.out"
value = 0 = 0x0
```

The following example shows how to initialize all the static objects that are currently loaded, with a single invocation of **cplusCtors()**:

```
-> cplusCtors
value = 0 = 0x0
```

---

**WARNING:** **cplusCtors()** should only be called once per module otherwise unpredictable behavior may result.

---

**RETURNS** N/A

**SEE ALSO** **cplusLib**, **cplusXtorSet()**

---

## cplusCtorsLink()

**NAME** cplusCtorsLink() – call all linked static constructors (C++)

**SYNOPSIS** `extern "C" void cplusCtorsLink ()`

**DESCRIPTION** This function calls constructors for all of the static objects linked with a VxWorks bootable image. When creating bootable applications, this function should be called from **usrRoot()** to initialize all static objects. Correct operation depends on correctly munching the C++ modules that are linked with VxWorks.

**RETURNS** N/A

**SEE ALSO** cplusLib

---

## cplusDemanglerSet()

**NAME** cplusDemanglerSet() – change C++ demangling mode (C++)

**SYNOPSIS** `extern "C" void cplusDemanglerSet  
(  
    int mode  
)`

**DESCRIPTION** This command sets the C++ demangling mode to *mode*. The default mode is 2.

There are three demangling modes, *complete*, *terse*, and *off*. These modes are represented by numeric codes:

| Mode     | Code |
|----------|------|
| off      | 0    |
| terse    | 1    |
| complete | 2    |

In complete mode, when C++ function names are printed, the class name (if any) is prefixed and the function's parameter type list is appended.

In terse mode, only the function name is printed. The class name and parameter type list are omitted.

In off mode, the function name is not demangled.

**EXAMPLES** The following example shows how one function name would be printed under each demangling mode:

| Mode     | Printed symbol                                             |
|----------|------------------------------------------------------------|
| off      | <code>_member__5classFPFI_PvPFPv_v</code>                  |
| terse    | <code>_member</code>                                       |
| complete | <code>foo::_member(void* (*)(long),void (*)(void*))</code> |

**RETURNS** N/A

**SEE ALSO** `cplusLib`

---

## cplusDemanglerStyleSet()

**NAME** `cplusDemanglerStyleSet()` – change C++ demangling style (C++)

**SYNOPSIS**

```
extern "C" void cplusDemanglerStyleSet
(
    DEMANGLER_STYLE style
)
```

**DESCRIPTION** This command sets the C++ demangling mode to *style*. The available demangler styles are enumerated in `demangler.h`. The default demangling style depends on the toolchain used to build the kernel. For example if the Diab toolchain is used to build the kernel then the default demangler style is `DMGL_STYLE_DIAB`.

**RETURNS** N/A

**SEE ALSO** `cplusLib`

---

## cplusDtors()

**NAME** `cplusDtors()` – call static destructors (C++)

**SYNOPSIS**

```
extern "C" void cplusDtors
(
    const char * moduleName
)
```

**DESCRIPTION** This function is used to call static destructors under the manual strategy (see **cplusXtorSet()**). *moduleName* is the name of an object module that was “munched” before loading. If *moduleName* is 0, then all static destructors, in all modules loaded by the VxWorks module loader, are called.

**EXAMPLES** The following example shows how to destroy the static objects in modules called “aplx.out” and “apply.out”:

```
-> cplusDtors "aplx.out"  
value = 0 = 0x0  
-> cplusDtors "apply.out"  
value = 0 = 0x0
```

The following example shows how to destroy all the static objects that are currently loaded, with a single invocation of **cplusDtors()**:

```
-> cplusDtors  
value = 0 = 0x0
```

---

**WARNING:** **cplusDtors()** should only be called once per module otherwise unpredictable behavior may result.

---

**RETURNS** N/A

**SEE ALSO** **cplusLib**, **cplusXtorSet()**

---

## **cplusDtorsLink()**

**NAME** **cplusDtorsLink()** – call all linked static destructors (C++)

**SYNOPSIS** **extern "C" void cplusDtorsLink ()**

**DESCRIPTION** This function calls destructors for all of the static objects linked with a VxWorks bootable image. When creating bootable applications, this function should be called during system shutdown to decommission all static objects. Correct operation depends on correctly munching the C++ modules that are linked with VxWorks.

**RETURNS** N/A

**SEE ALSO** **cplusLib**

---

## cplusLibInit()

**NAME** `cplusLibInit()` – initialize the C++ library (C++)

**SYNOPSIS** `extern "C" STATUS cplusLibInit (void)`

**DESCRIPTION** This routine initializes the C++ library and forces all C++ run-time support to be linked with the bootable VxWorks image. If the configuration macro `INCLUDE_CPLUS` is defined, `cplusLibInit()` is called automatically from the root task, `usrRoot()`, in `usrConfig.c`.

**RETURNS** OK or ERROR.

**SEE ALSO** `cplusLib`

---

## cplusXtorSet()

**NAME** `cplusXtorSet()` – change C++ static constructor calling strategy (C++)

**SYNOPSIS** `extern "C" void cplusXtorSet  
(  
    int strategy  
)`

**DESCRIPTION** This command sets the C++ static constructor calling strategy to *strategy*. The default strategy is 1.

There are two static constructor calling strategies: *automatic* and *manual*. These modes are represented by numeric codes:

| Strategy  | Code |
|-----------|------|
| manual    | 0    |
| automatic | 1    |

Under the manual strategy, a module's static constructors and destructors are called by `cplusCtors()` and `cplusDtors()`, which are themselves invoked manually.

Under the automatic strategy, a module's static constructors are called as a side-effect of loading the module using the VxWorks module loader. A module's static destructors are called as a side-effect of unloading the module.

**cpsr()**


---

**NOTE:** The manual strategy is applicable only to modules that are loaded by the VxWorks module loader. Static constructors and destructors contained by modules linked with the VxWorks image are called using **cplusCtorsLink()** and **cplusDtorsLink()**.

---

**RETURNS** N/A

**SEE ALSO** **cplusLib**

---

## **cpsr()**

**NAME** **cpsr()** – return the contents of the current processor status register (ARM)

**SYNOPSIS**

```
int cpsr
(
    int taskId           /* task ID, 0 means default task */
)
```

**DESCRIPTION** This command extracts the contents of the status register from the TCB of a specified task. If *taskId* is omitted or zero, the last task referenced is assumed.

**RETURNS** The contents of the current processor status register.

**SEE ALSO** **dbgArchLib**, *VxWorks Programmer's Guide: Debugging*

---

## **creat()**

**NAME** **creat()** – create a file

**SYNOPSIS**

```
int creat
(
    const char * name,    /* name of the file to create */
    int         flag,     /* O_RDONLY, O_WRONLY, or O_RDWR */
)
```

**DESCRIPTION** This routine creates a file called *name* and opens it with a specified *flag*. This routine determines on which device to create the file; it then calls the create routine of the device driver to do most of the work. Therefore, much of what transpires is device/driver-dependent.

The parameter *flag* is set to **O\_RDONLY** (0), **O\_WRONLY** (1), or **O\_RDWR** (2) for the duration of time the file is open. To create NFS files with a UNIX `chmod`-type file mode, call **open()** with the file mode specified in the third argument.

---

**NOTE:** For more information about situations when there are no file descriptors available, see the manual entry for **iosInit()**.

---

**RETURNS** A file descriptor number, or **ERROR** if a filename is not specified, the device does not exist, no file descriptors are available, or the driver returns **ERROR**.

**SEE ALSO** **ioLib**, **open()**

---

## cret()

**NAME** **cret()** – continue until the current subroutine returns

**SYNOPSIS**

```
STATUS cret
(
    int task                /* task to continue, 0 = default */
)
```

**DESCRIPTION** This routine places a breakpoint at the return address of the current subroutine of a specified task, then continues execution of that task.

To execute, enter:

```
-> cret [task]
```

If *task* is omitted or zero, the last task referenced is assumed.

When the breakpoint is hit, information about the task will be printed in the same format as in single-stepping. The breakpoint is automatically removed when hit, or if the task hits another breakpoint first.

**RETURNS** **OK**, or **ERROR** if there is no such task or the breakpoint table is full.

**SEE ALSO** **dbgLib**, **so()**, *VxWorks Programmer's Guide: Target Shell*, **windsh**, *Tornado User's Guide: Shell*

**ctime()**

**NAME** `ctime()` – convert time in seconds into a string (ANSI)

**SYNOPSIS**

```
char * ctime
(
    const time_t * timer    /* calendar time in seconds */
)
```

**DESCRIPTION** This routine converts the calendar time pointed to by *timer* into local time in the form of a string. It is equivalent to:

```
asctime (localtime (timer));
```

This routine is not reentrant. For a reentrant version, see `ctime_r()`.

**INCLUDE FILES** `time.h`

**RETURNS** The pointer returned by `asctime()` with local broken-down time as the argument.

**SEE ALSO** `ansiTime`, `asctime()`, `localtime()`

**ctime\_r()**

**NAME** `ctime_r()` – convert time in seconds into a string (POSIX)

**SYNOPSIS**

```
char * ctime_r
(
    const time_t * timer,    /* calendar time in seconds */
    char *        asctimeBuf, /* buffer to contain the string */
    size_t *      buflen     /* size of the buffer */
)
```

**DESCRIPTION** This routine converts the calendar time pointed to by *timer* into local time in the form of a string. It is equivalent to:

```
asctime (localtime (timer));
```

This routine is the POSIX re-entrant version of `ctime()`.

**INCLUDE FILES** `time.h`

**RETURNS** The pointer returned by **asctime()** with local broken-down time as the argument.

**SEE ALSO** **ansiTime, asctime(), localtime()**

**d()****NAME** d() – display memory

**SYNOPSIS**

```
void d
(
    void * adrs,      /* address to display (if 0, display next block */
    int   nunits,    /* number of units to print (if 0, use default) */
    int   width      /* width of displaying unit (1, 2, 4, 8)      */
)
```

**DESCRIPTION** This command displays the contents of memory, starting at *adrs*. If *adrs* is omitted or zero, **d()** displays the next memory block, starting from where the last **d()** command completed.

Memory is displayed in units specified by *width*. If *nunits* is omitted or zero, the number of units displayed defaults to last use. If *nunits* is non-zero, that number of units is displayed and that number then becomes the default. If *width* is omitted or zero, it defaults to the previous value. If *width* is an invalid number, it is set to 1. The valid values for *width* are 1, 2, 4, and 8. The number of units **d()** displays is rounded up to the nearest number of full lines.

**RETURNS** N/A

**SEE ALSO** **usrLib**, **m()**, *VxWorks Programmer's Guide: Target Shell*, **windsh**, *Tornado User's Guide: Shell*

**d0()****NAME** d0() – return the contents of register d0 (also d1 - d7) (68K)

**SYNOPSIS**

```
int d0
(
    int taskId        /* task ID, 0 means default task */
)
```

**DESCRIPTION** This command extracts the contents of register **d0** from the TCB of a specified task. If *taskId* is omitted or zero, the last task referenced is assumed.

Similar routines are provided for all data registers (**d0 - d7**): **d0()** - **d7()**.

**RETURNS** The contents of register **d0** (or the requested register).

**SEE ALSO** **dbgArchLib**, *VxWorks Programmer's Guide: Target Shell*

---

## dbgBpTypeBind()

**NAME** `dbgBpTypeBind()` – bind a breakpoint handler to a breakpoint type (MIPS)

**SYNOPSIS**

```
STATUS dbgBpTypeBind
(
    int      bpType,          /* breakpoint type */
    FUNCPTR  routine         /* function to bind */
)
```

**DESCRIPTION** Dynamically bind a breakpoint handler to breakpoints of type 0 - 7. By default only breakpoints of type zero are handled with the vxWorks breakpoint handler (see `dbgLib`). Other types may be used for Ada stack overflow or other such functions. The installed handler must take the same parameters as `excExcHandle()` (see `excLib`).

**RETURNS** OK, or ERROR if *bpType* is out of bounds.

**SEE ALSO** `dbgArchLib`, `dbgLib`, `excLib`

---

## dbgHelp()

**NAME** `dbgHelp()` – display debugging help menu

**SYNOPSIS** `void dbgHelp (void)`

**DESCRIPTION** This routine displays a summary of `dbgLib` utilities with a short description of each, similar to the following:

|                      |                                                 |                                  |
|----------------------|-------------------------------------------------|----------------------------------|
| <code>dbgHelp</code> |                                                 | Print this list                  |
| <code>dbgInit</code> |                                                 | Install debug facilities         |
| <code>b</code>       |                                                 | Display breakpoints              |
| <code>b</code>       | <code>addr[,task[,count]]</code>                | Set breakpoint                   |
| <code>e</code>       | <code>addr[,eventNo[,task[,func[,arg]]]]</code> | Set eventpoint (WindView)        |
| <code>bd</code>      | <code>addr[,task]</code>                        | Delete breakpoint                |
| <code>bdall</code>   | <code>[task]</code>                             | Delete all breakpoints           |
| <code>c</code>       | <code>[task[,addr[,addr1]]]</code>              | Continue from breakpoint         |
| <code>cret</code>    | <code>[task]</code>                             | Continue to subroutine return    |
| <code>s</code>       | <code>[task[,addr[,addr1]]]</code>              | Single step                      |
| <code>so</code>      | <code>[task]</code>                             | Single step/step over subroutine |
| <code>l</code>       | <code>[adr[,nInst]]</code>                      | List disassembled memory         |
| <code>tt</code>      | <code>[task]</code>                             | Do stack trace on task           |

**dbgInit()**

```

    bh          addr[,access[,task[,count[,quiet]]]] set hardware breakpoint
                (if supported by the architecture)

```

**RETURNS** N/A

**SEE ALSO** **dbgLib**, *VxWorks Programmer's Guide: Target Shell*

**dbgInit()**

**NAME** **dbgInit()** – initialize the local debugging package

**SYNOPSIS** **STATUS** **dbgInit** (void)

**DESCRIPTION** This routine initializes the local debugging package and enables the basic breakpoint and single-step functions.

This routine also enables the shell abort function, CTRL-C.

**NOTE:** The debugging package should be initialized before any debugging routines are used. If the configuration macro **INCLUDE\_DEBUG** is defined, **dbgInit()** is called by the root task, **usrRoot()**, in **usrConfig.c**.

**RETURNS** OK, always.

**SEE ALSO** **dbgLib**, *VxWorks Programmer's Guide: Target Shell*

**dcacheDevCreate()**

**NAME** **dcacheDevCreate()** – create a disk cache

**SYNOPSIS** **CBIO\_DEV\_ID** **dcacheDevCreate**

```

    (
    CBIO_DEV_ID subDev,      /* block device handle */
    char *      pRamAddr,   /* where it is in memory (NULL = KHEAP_ALLOC) */
    int         memSize,    /* amount of memory to use */
    char *      pDesc       /* device description string */
    )

```

- DESCRIPTION** This routine creates a CBIO layer disk data cache instance. The disk cache unit accesses the disk through the subordinate CBIO device driver, provided with the *subDev* argument.
- A valid block device **BLK\_DEV** handle may be provided instead of a CBIO handle, in which case it will be automatically converted into a CBIO device by using the wrapper functionality from **cbioLib**.
- Memory which will be used for caching disk data may be provided by the caller with *pRamAddr*, or it will be allocated by **dcacheDevCreate()** from the common system memory pool, if *memAddr* is passed as **NULL**. *memSize* is the amount of memory to use for disk caching, if 0 is passed, then a certain default value will be calculated, based on available memory. *pDesc* is a string describing the device, used later by **dcacheShow()**, and is useful when there are many cached disk devices.
- A maximum of 16 disk cache devices are supported at this time.
- RETURNS** disk cache device handle, or **NULL** if there is not enough memory to satisfy the request, or the *blkDev* handle is invalid.
- SEE ALSO** **dcacheCbio**

---

## dcacheDevDisable()

- NAME** **dcacheDevDisable()** – disable the disk cache for this device
- SYNOPSIS**
- ```
STATUS dcacheDevDisable
(
    CBIO_DEV_ID dev          /* CBIO device handle */
)
```
- DESCRIPTION** This function disables the cache by setting the bypass count to zero and storing the old value, if there is already an old value then we won't repeat the process though.
- RETURNS** **OK** if cache is successfully disabled or **ERROR**.
- SEE ALSO** **dcacheCbio**

## dcacheDevEnable()

<b>NAME</b>	<b>dcacheDevEnable()</b> – re-enable the disk cache
<b>SYNOPSIS</b>	<pre><b>STATUS</b> dcacheDevEnable (     <b>CBIO_DEV_ID</b> dev          /* CBIO device handle */ )</pre>
<b>DESCRIPTION</b>	This function re-enables the cache if we disabled it. If we did not disable it, then we cannot re-enable it.
<b>RETURNS</b>	OK if cache is successfully enabled or <b>ERROR</b> .
<b>SEE ALSO</b>	<b>dcacheCbio</b>

---

## dcacheDevMemResize()

<b>NAME</b>	<b>dcacheDevMemResize()</b> – set a new size to a disk cache device
<b>SYNOPSIS</b>	<pre><b>STATUS</b> dcacheDevMemResize (     <b>CBIO_DEV_ID</b> dev,        /* device handle */     <b>size_t</b>    newSize     /* new cache size in bytes */ )</pre>
<b>DESCRIPTION</b>	This routine is used to resize the dcache layer. This routine is also useful after a disk change event, for example a PCMCIA disk swap. The routine <b>pccardDosDevCreate()</b> in <b>pccardLib.c</b> uses this routine for that function. This should be invoked each time a new disk is inserted on media where the device geometry could possibly change. This function will re-read all device geometry data from the block driver, carve out and initialize all cache descriptors and blocks.
<b>RETURNS</b>	OK or <b>ERROR</b> if the device is invalid or if the device geometry is invalid ( <b>EINVAL</b> ) or if there is not enough memory to perform the operation.
<b>SEE ALSO</b>	<b>dcacheCbio</b>

---

## dcacheDevTune()

**NAME** `dcacheDevTune()` – modify tunable disk cache parameters

**SYNOPSIS**

```
STATUS dcacheDevTune
(
    CBIO_DEV_ID dev,          /* device handle */
    int dirtyMax,           /* max # of dirty cache blocks allowed */
    int bypassCount,       /* request size for bypassing cache */
    int readAhead,         /* how many blocks to read ahead */
    int syncInterval      /* how many seconds between disk updates */
)
```

**DESCRIPTION** This function allows the user to tune some disk cache parameters to obtain better performance for a given application or workload pattern. These parameters are checked for sanity before being used, hence it is recommended to verify the actual parameters being set with `dcacheShow()`.

Following is the description of each tunable parameter:

### **bypassCount**

In order to achieve maximum performance, Disk Cache is bypassed for very large requests. This parameter sets the threshold number of blocks for bypassing the cache, resulting usually in the data being transferred by the low level driver directly to/from application data buffers (also known as cut-through DMA). Passing the value of 0 in this argument preserves the previous value of the associated parameter.

### **syncInterval**

The Disk Cache provides a low priority task that will update all modified blocks onto the disk periodically. This parameters controls the time between these updates in seconds. The longer this period, the better throughput is likely to be achieved, while risking to loose more data in the event of a failure. For removable devices this interval is fixed at 1 second. Setting this parameter to 0 results in immediate writes to disk when requested, resulting in minimal data loss risk at the cost of somewhat degraded performance.

### **readAhead**

In order to avoid accessing the disk in small units, the Disk Cache will read many contiguous blocks once a block which is absent from the cache is needed. Increasing this value increases read performance, but a value which is too large may cause blocks which are frequently used to be removed from the cache, resulting in a low Hit Ratio, and increasing the number of Seeks, slowing down performance dramatically. Passing the value of 0 in this argument preserves the pervious value of the associated parameter.

***dcacheHashTest()*****dirtyMax**

Routinely the Disk Cache will keep modified blocks in memory until it is specifically instructed to update these blocks to the disk, or until the specified time interval between disk updates has elapsed, or until the number of modified blocks is large enough to justify an update. Because the disk is updated in an ordered manner, and the blocks are written in groups when adjacent blocks have been modified, a larger *dirtyMax* parameter will minimize the number of Seek operation, but a value which is too large may decrease the Hit Ratio, thus degrading performance. Passing the value of 0 in this argument preserves the pervious value of the associated parameter.

**RETURNS** OK or ERROR if device handle is invalid. Parameter value which is out of range will be silently corrected.

**SEE ALSO** `dcacheCbio`, `dcacheShow()`

---

## **dcacheHashTest()**

**NAME** `dcacheHashTest()` – test hash table integrity

**SYNOPSIS**

```
void dcacheHashTest
(
    CBIO_DEV_ID dev
)
```

**DESCRIPTION**

**SEE ALSO** `dcacheCbio`

---

## dcacheShow()

**NAME** `dcacheShow()` – print information about disk cache

**SYNOPSIS**

```
void dcacheShow
(
    CBIO_DEV_ID dev,          /* device handle */
    int         verbose      /* 1 - display state of each cache block */
)
```

**DESCRIPTION** This routine displays various information regarding a disk cache, namely current disk parameters, cache size, tunable parameters and performance statistics. The information is displayed on the standard output.

The *dev* argument is the device handle, if it is **NULL**, all disk caches are displayed.

**RETURNS** N/A

**SEE ALSO** `dcacheCbio`

---

## devs()

**NAME** `devs()` – list all system-known devices

**SYNOPSIS** `void devs (void)`

**DESCRIPTION** This command displays a list of all devices known to the I/O system.

**RETURNS** N/A

**SEE ALSO** `usrLib`, `iosDevShow()`, *VxWorks Programmer's Guide: Target Shell*, `windsh`, *Tornado User's Guide: Shell*

---

## dhcpcBind()

<b>NAME</b>	<b>dhcpcBind()</b> – obtain a set of network configuration parameters with DHCP
<b>SYNOPSIS</b>	<pre><b>STATUS</b> dhcpcBind (     void * pCookie,          /* identifier returned by dhcpcInit() */     <b>BOOL</b>  syncFlag        /* synchronous or asynchronous execution */ )</pre>
<b>DESCRIPTION</b>	<p>This routine initiates a DHCP negotiation according to the process described in RFC 1541. The <i>pCookie</i> argument contains the return value of an earlier <b>dhcpcInit()</b> call and is used to identify a particular lease.</p> <p>The <i>syncFlag</i> parameter specifies whether the DHCP negotiation started by this routine will execute synchronously or asynchronously. An asynchronous execution will return after starting the DHCP negotiation, but a synchronous execution will only return once the negotiation process completes.</p> <p>When a new lease is established, any event hook provided for the lease will be called to process the configuration parameters. The hook is also called when the lease expires or the negotiation process fails. The results of an asynchronous DHCP negotiation are not available unless an event hook is installed.</p> <p>If automatic configuration of the underlying network interface was specified during the lease initialization, this routine will prevent all higher-level protocols from accessing the underlying network interface used during the initial lease negotiation until that process is complete. In addition, any addressing information obtained will be applied to that network interface, which will remain disabled if the initial negotiation fails. Finally, the interface will be disabled if the lease expires.</p> <hr/> <p><b>NOTE:</b> If the DHCP client is used to obtain the VxWorks boot parameters, this routine is called automatically during system startup using the automatic reconfiguration. Therefore, any calls to this routine which use the network boot device for message transfer when the DHCP client was used at boot time must not request automatic reconfiguration during initialization. Otherwise, the resulting lease settings will conflict with the configuration maintained by the lease established during system startup.</p> <hr/>
<b>RETURNS</b>	OK if routine completes, or <b>ERROR</b> otherwise.
<b>ERRNO</b>	S_dhcpcLib_BAD_COOKIE, S_dhcpcLib_NOT_INITIALIZED, S_dhcpcLib_BAD_OPTION, S_dhcpcLib_BAD_DEVICE
<b>SEE ALSO</b>	<b>dhcpcLib</b>

---

## dhcpcBootBind()

**NAME** `dhcpcBootBind()` – initialize the network with DHCP at boot time

**SYNOPSIS** `STATUS dhcpcBootBind (void)`

**DESCRIPTION** This routine performs the client side of a DHCP negotiation according to RFC 2131. The negotiation uses the network device specified with the initialization call. The addressing information retrieved is applied to that network device. Because the boot image is replaced by the downloaded target image, the resulting lease cannot be renewed. Therefore, the minimum lease length specified by `DHCPC_MIN_LEASE` must be set so that the target image has sufficient time to download and begin monitoring the lease. This routine is called automatically by the boot program when `INCLUDE_DHCP` is defined and the automatic configuration option is set in the boot flags and no target address is present.

**RETURNS** `OK` if negotiation is successful, or `ERROR` otherwise.

**ERRNO** N/A

**SEE ALSO** `dhcpcBootLib`

---

## dhcpcBootInformGet()

**NAME** `dhcpcBootInformGet()` – obtain additional configuration parameters with DHCP

**SYNOPSIS** `STATUS dhcpcBootInformGet`  
`(`  
`char * pAddrString           /* known address assigned to client */`  
`)`

**DESCRIPTION** This routine uses DHCP to retrieve additional configuration parameters for a client with the externally configured network address given by the *pAddrString* parameter. It sends an INFORM message and waits for a reply following the process described in RFC 2131. The message exchange uses the network device specified with the initialization call. Any interface information retrieved is applied to that network device. Since this process does not establish a lease, the target address will not contain any timestamp information so that the runtime image will not attempt to verify the configuration parameters. This routine is called automatically by the boot program when `INCLUDE_DHCP` is defined and the automatic configuration option is set in the boot flags if a target address is already present.

**RETURNS** OK if negotiation is successful, or **ERROR** otherwise.

**ERRNO** N/A

**SEE ALSO** **dhcpcBootLib**

---

## dhcpcBootInit()

**NAME** **dhcpcBootInit()** – set up the DHCP client parameters and data structures

**SYNOPSIS**

```
void * dhcpcBootInit
(
    struct ifnet * pIf,      /* network device used by client */
    int          serverPort, /* port used by DHCP servers (default 67) */
    int          clientPort, /* port used by DHCP clients (default 68) */
    int          maxSize,    /* largest DHCP message supported, in bytes */
    int          offerTimeout, /* interval to get additional DHCP offers */
    int          defaultLease, /* default value for requested lease length */
    int          minLease    /* minimum accepted lease length */
)
```

**DESCRIPTION** This routine creates any necessary data structures and sets the client's option request list to retrieve a subnet mask and broadcast address for the network interface indicated by *pIf*. The routine is executed automatically by the boot program when **INCLUDE\_DHCP** is defined and the automatic configuration option is set in the boot flags. The network interface specified by *pIf* is used to transmit and receive all DHCP messages during the lease negotiation. The DHCP client supports interfaces attached to the IP protocol using the MUX/END interface and BSD Ethernet devices attached to that protocol. The interface must be capable of sending broadcast messages. The *maxSize* parameter specifies the maximum length supported for any DHCP message, including the UDP and IP headers and the link level header. The maximum length of the DHCP options field is based on this value or the MTU size for the given interface, whichever is less. The smallest valid value for the *maxSize* parameter is 576 bytes, corresponding to the minimum IP datagram a host must accept. The MTU size of the network interface must be large enough to handle those datagrams.

**ERRNO** N/A

**RETURNS** Lease handle for later use, or **NULL** if lease startup fails.

**SEE ALSO** **dhcpcBootLib**

---

## dhcpcCacheHookAdd()

**NAME** `dhcpcCacheHookAdd()` – add a routine to store and retrieve lease data

**SYNOPSIS**

```
STATUS dhcpcCacheHookAdd
(
    FUNCPTR pCacheHookRtn    /* routine to store/retrieve lease data */
)
```

**DESCRIPTION** This routine adds a hook routine that is called at the bound state (to store the lease data) and during the `INIT_REBOOT` state (to re-use the parameters if the lease is still active). The calling sequence of the input hook routine is:

```
STATUS dhcpcCacheHookRtn
(
    int command,                /* requested cache operation */
    unsigned long *pTimeStamp,  /* lease timestamp data */
    int *pDataLen,             /* length of data to access */
    char *pBuffer              /* pointer to data buffer */
)
```

The hook routine should return `OK` if the requested operation is completed successfully, or `ERROR` otherwise. All the supplied pointers reference memory locations that are reused upon return from the hook. The hook routine must copy the data elsewhere.

---

**NOTE:** The setting of the cache hook routine during a `dhcpcInit()` call is recorded and used by the resulting lease throughout its lifetime. Since the hook routine is intended to store a single lease record, a separate hook routine should be specified before the `dhcpcInit()` call for each lease which will re-use its parameters across reboots.

---

**IMPLEMENTATION** The *command* parameter specifies one of the following operations:

### `DHCP_CACHE_WRITE`

Save the indicated data. The write operation must preserve the value referenced by *pTimeStamp* and the contents of *pBuffer*. The *pDataLen* parameter indicates the number of bytes in that buffer.

### `DHCP_CACHE_READ`

Restore the saved data. The read operation must copy the data from the most recent write operation into the location indicated by *pBuffer*, set the contents of *pDataLen* to the amount of data provided, and store the corresponding timestamp value in *pTimeStamp*.

- The read operation has very specific requirements. On entry, the value referenced by *pDataLen* indicates the maximum buffer size available at *pBuffer*. If the amount of data stored by the previous write exceeds this value, the operation must return `ERROR`. A

read must also return **ERROR** if the saved timestamp value is 0. Finally, the read operation must return **ERROR** if it is unable to retrieve all the data stored by the write operation or if the previous write was unsuccessful.

**DHCP\_CACHE\_ERASE**

Ignore all stored data. Following this operation, subsequent read operations must return **ERROR** until new data is written. All parameters except *command* are **NULL**.

**RETURNS** OK, always.

**ERRNO** N/A

**SEE ALSO** **dhcpcLib**

---

## **dhcpcCacheHookDelete()**

**NAME** **dhcpcCacheHookDelete()** – delete a lease data storage routine

**SYNOPSIS** **STATUS dhcpcCacheHookDelete (void)**

**DESCRIPTION** This routine deletes the hook used to store lease data, preventing re-use of the configuration parameters across system reboots for all subsequent lease attempts. Currently active leases will continue to use the routine specified before the lease initialization.

**RETURNS** OK, always.

**ERRNO** N/A

**SEE ALSO** **dhcpcLib**

---

## dhcpcEventHookAdd()

**NAME** `dhcpcEventHookAdd()` – add a routine to handle configuration parameters

**SYNOPSIS**

```
STATUS dhcpcEventHookAdd
(
    void * pCookie,          /* identifier returned by dhcpcInit() */
    FUNCPTR pEventHook      /* routine to handle lease parameters */
)
```

**DESCRIPTION** This routine installs a hook routine to handle changes in the configuration parameters provided for the lease indicated by *pCookie*. The hook provides an alternate configuration method for DHCP leases and uses the following interface:

```
void dhcpcEventHookRtn
(
    int      leaseEvent,    /* new or expired parameters */
    void *   pCookie       /* lease identifier from dhcpcInit() */
)
```

The routine is called with the *leaseEvent* parameter set to `DHCPC_LEASE_NEW` whenever a lease is successfully established. The `DHCPC_LEASE_NEW` event does not occur when a lease is renewed by the same DHCP server, since the parameters do not change in that case. However, it does occur if the client rebinds to a different DHCP server. The `DHCPC_LEASE_INVALID` event indicates that the configuration parameters for the corresponding lease may no longer be used. That event occurs when a lease expires or a renewal or verification attempt fails, and coincides with re-entry into the initial state of the negotiation process.

If the lease initialization specified automatic configuration of the corresponding network interface, any installed hook routine will be invoked after the new address information is applied.

**RETURNS** `OK` if notification hook added, or `ERROR` otherwise.

**ERRNO** `S_dhcpcLib_BAD_COOKIE`, `S_dhcpcLib_NOT_INITIALIZED`

**SEE ALSO** `dhcpcLib`

## **dhcpcEventHookDelete()**

- NAME** `dhcpcEventHookDelete()` – remove the configuration parameters handler
- SYNOPSIS**
- ```
STATUS dhcpcEventHookDelete  
(  
    void * pCookie          /* identifier returned by dhcpcInit() */  
)
```
- DESCRIPTION** This routine removes the hook routine that handled changes in the configuration parameters for the lease indicated by *pCookie*. If the lease initialization specified automatic configuration of the corresponding network interface, the assigned address could change without warning after this routine is executed.
- RETURNS** OK if notification hook removed, or **ERROR** otherwise.
- ERRNO** `S_dhcpcLib_BAD_COOKIE`, `S_dhcpcLib_NOT_INITIALIZED`
- SEE ALSO** `dhcpcLib`
- 

## **dhcpcInformGet()**

- NAME** `dhcpcInformGet()` – obtain additional configuration parameters with DHCP
- SYNOPSIS**
- ```
STATUS dhcpcInformGet  
(  
    void * pCookie,          /* identifier returned by dhcpcInit() */  
    char * pAddrString,     /* known address assigned to client */  
    BOOL  syncFlag         /* synchronous or asynchronous execution? */  
)
```
- DESCRIPTION** This routine uses DHCP to retrieve additional configuration parameters for a client with the externally configured network address given by the *pAddrString* parameter. It sends an INFORM message and waits for a reply following the process described in RFC 2131. The *pCookie* argument contains the return value of an earlier **dhcpcInit()** call and is used to access the resulting configuration. Unlike the **dhcpcBind()** call, this routine does not establish a lease with a server.
- The *syncFlag* parameter specifies whether the message exchange started by this routine will execute synchronously or asynchronously. An asynchronous execution will return

after sending the initial message, but a synchronous execution will only return once the process completes.

When a server responds with an acknowledgement message, any event hook provided will be called to process the configuration parameters. The hook is also called if the message exchange fails. The results of an asynchronous execution are not available unless an event hook is installed.

---

**NOTE:** This routine is designed as an alternative to the **dhcpcBind()** routine. It should not be used for any **dhcpcInit()** identifier corresponding to an active or pending lease.

---

**RETURNS** OK if routine completes, or **ERROR** otherwise.

**ERRNO** S\_dhcpcLib\_BAD\_COOKIE, S\_dhcpcLib\_NOT\_INITIALIZED, S\_dhcpcLib\_BAD\_OPTION

**SEE ALSO** dhcpcLib

---

## dhcpcInit()

**NAME** dhcpcInit() – assign network interface and setup lease request

**SYNOPSIS**

```
void * dhcpcInit
(
    struct ifnet * pIf,          /* network device used by client */
    BOOL          autoConfig /* reconfigure network device? */
)
```

**DESCRIPTION** This routine creates the data structures used to obtain a set of parameters with DHCP and must be called before each attempt at establishing a DHCP lease, but after the **dhcpcLibInit()** routine has initialized the global data structures.

The *pIf* argument indicates the network device which will be used for transmission and reception of DHCP messages during the lifetime of the lease. The DHCP client supports devices attached to the IP protocol with the MUX/END interface. The specified device must be capable of sending broadcast messages. It also supports BSD Ethernet devices attached to the IP protocol. The MTU size of any interface must be large enough to receive a minimum IP datagram of 576 bytes. If the interface MTU size is less than the maximum message size set in the library initialization it also determines the maximum length of the DHCP options field.

If the *autoConfig* parameter is set to **TRUE**, any address information obtained will automatically be applied to the specified interface. The *autoConfig* parameter also selects the default option request list for a lease. If set to **FALSE**, no specific lease options are

requested since any configuration parameters obtained are not intended for the underlying network device. In that case, any specific options required may be added to the request list at any time before the corresponding **dhcpcBind()** call. If *autoConfig* is **TRUE**, this routine sets the configuration parameters to request the minimal address information (subnet mask and broadcast address) necessary for reconfiguring the network device specified by *plf*.

The internal lease identifier returned by this routine must be used in subsequent calls to the DHCP client library.

---

**NOTE:** This routine is called automatically during system startup if the DHCP client was used to obtain the VxWorks boot parameters. The resulting lease will always reconfigure the network boot device. Therefore, any further calls to this routine which specify the network boot device for use in obtaining additional DHCP leases must set *autoConfig* to **FALSE**. Otherwise, that device will be unable to maintain a stable configuration. The global variable **pDhcpBootCookie** provides access to the configuration parameters for any DHCP lease created during system startup.

---

<b>RETURNS</b>	Lease handle for later use, or <b>NULL</b> if lease setup fails.
<b>ERRNO</b>	<b>S_dhcpcLib_NOT_INITIALIZED</b> , <b>S_dhcpcLib_BAD_DEVICE</b> , <b>S_dhcpcLib_BAD_OPTION</b> , <b>S_dhcpcLib_MAX_LEASES_REACHED</b> , <b>S_dhcpcLib_MEM_ERROR</b>
<b>SEE ALSO</b>	<b>dhcpcLib</b> , <b>dhcpcOptionSet()</b> , <b>dhcpcEventHookAdd()</b>

---

## dhcpcLibInit()

**NAME** **dhcpcLibInit()** – DHCP client library initialization

**SYNOPSIS**

```
STATUS dhcpcLibInit
(
    int serverPort,          /* port used by DHCP servers (default 67) */
    int clientPort,        /* port used by DHCP clients (default 68) */
    int maxLeases,         /* max number of simultaneous leases allowed */
    int maxSize,           /* largest DHCP message supported, in bytes */
    int offerTimeout,      /* interval to get additional DHCP offers */
    int defaultLease,      /* default value for requested lease length */
    int minLease           /* minimum accepted lease length */
)
```

**DESCRIPTION** This routine creates and initializes the global data structures used by the DHCP client library to maintain multiple leases, up to the limit specified by the *maxLeases* parameter.

Every subsequent lease attempt will collect additional DHCP offers until the interval specified by *offerTimeout* expires and will request the lease duration indicated by *defaultLease*. The *maxSize* parameter specifies the maximum length supported for any DHCP message, including the UDP and IP headers and the largest link level header for all supported devices. The maximum length of the DHCP options field is based on this value or the MTU size for a lease's underlying interface, whichever is less. The smallest valid value for the *maxSize* parameter is 576 bytes, corresponding to the minimum IP datagram a host must accept. Larger values will allow the client to handle longer DHCP messages.

This routine must be called before calling any other library routines. The routine is called automatically if `INCLUDE_DHCP` is defined at the time the system is built and assigns the global lease settings to the values specified by `DHCPC_SPORT`, `DHCPC_CPORT`, `DHCPC_MAX_LEASES`, `DHCPC_MAX_MSGSIZE`, `DHCPC_DEFAULT_LEASE`, and `DHCPC_OFFER_TIMEOUT`.

**RETURNS** OK, or `ERROR` if initialization fails.

**ERRNO** `S_dhcpcLib_MEM_ERROR`

**SEE ALSO** `dhcpcLib`

---

## dhcpcOptionAdd()

**NAME** `dhcpcOptionAdd()` – add an option to the client messages

**SYNOPSIS**

```
STATUS dhcpcOptionAdd
(
    void * pCookie,           /* identifier returned by dhcpcInit() */
    UCHAR option,           /* RFC 2132 tag of desired option */
    int length,             /* length of option data */
    UCHAR * pData           /* option data */
)
```

**DESCRIPTION** This routine inserts option tags and associated values into the body of all outgoing messages for the lease indicated by the *pCookie* parameter. Each lease can accept option data up to the MTU size of the underlying interface, minus the link-level header size and the additional 283 bytes required for a minimum DHCP message (including mandatory options).

The *option* parameter specifies an option tag defined in RFC 2132. See the `dhcp/dhcp.h` include file for a listing of defined aliases for the available option tags. This routine will not accept the following *option* values, which are used for control purposes and cannot be included arbitrarily:

```
_DHCP_PAD_TAG  
_DHCP_OPT_OVERLOAD_TAG  
_DHCP_MSGTYPE_TAG  
_DHCP_SERVER_ID_TAG  
_DHCP_MAXMSGSIZE_TAG  
_DHCP_END_TAG
```

This routine also will not accept *option* values 62 or 63, which are not currently defined.

The *length* parameter indicates the number of bytes in the option body provided by the *pData* parameter.

The maximum length of the option field in a DHCP message depends on the MTU size of the associated interface and the maximum DHCP message size set during the DHCP library initialization. These option settings share that field with any option request list created through the **dhcpcOptionSet()** routine. Options which exceed the limit will not be stored.

Each call to this routine with the same *option* value usually replaces the value of the existing option, if any. However, the routine will append the new data for the *option* values which contain variable length lists, corresponding to tags 3-11, 21, 25, 33, 41-45, 48-49, 55, 65, and 68-76.

---

**WARNING:** The `_DHCP_REQ_LIST_TAG` *option* value (55) will replace any existing list created with the **dhcpcOptionSet()** routine.

---

<b>RETURNS</b>	OK if the option was inserted successfully, or <b>ERROR</b> if the option is invalid or storage failed.
<b>ERRNO</b>	S_dhcpcLib_BAD_OPTION, S_dhcpcLib_OPTION_NOT_STORED
<b>SEE ALSO</b>	dhcpcCommonLib

---

## dhcpcOptionGet()

**NAME** dhcpcOptionGet() – retrieve an option provided to a client and store in a buffer

**SYNOPSIS**

```
STATUS dhcpcOptionGet  
(  
    void * pCookie,          /* identifier returned by dhcpcInit() */  
    int   option,           /* RFC 2132 option tag */  
    int * pLength,          /* size of provided buffer and data returned */  
    char * pBuf             /* location for option data */  
)
```

**DESCRIPTION** This routine retrieves the data for a specified option from a lease indicated by the *pCookie* parameter. The *option* parameter specifies an option tag as defined in RFC 2132. See the **dhcpc/dhcp.h** include file for a listing of defined aliases for the available option tags. This routine will not accept the following *option* values, which are either used by the server for control purposes or only supplied by the client:

```

_DHCP_PAD_TAG
_DHCP_REQUEST_IPADDR_TAG
_DHCP_OPT_OVERLOAD_TAG
_DHCP_MSGTYPE_TAG
_DHCP_REQ_LIST_TAG
_DHCP_MAXMSGSIZE_TAG
_DHCP_CLASS_ID_TAG
_DHCP_CLIENT_ID_TAG
_DHCP_END_TAG

```

If the option is found, the data is stored in the provided buffer, up to the limit specified in the *pLength* parameter. The option is not available if the DHCP client is not in the bound state or if the server did not provide it. After returning, the *pLength* parameter indicates the amount of data actually retrieved. The provided buffer may contain IP addresses stored in network byte order. All other numeric values are stored in host byte order. See RFC 2132 for specific details on the data retrieved.

**RETURNS** OK if option available, or **ERROR** otherwise.

**ERRNO** S\_dhcpcLib\_BAD\_COOKIE, S\_dhcpcLib\_NOT\_INITIALIZED, S\_dhcpcLib\_NOT\_BOUND, S\_dhcpcLib\_OPTION\_NOT\_PRESENT

**SEE ALSO** dhcpcLib, dhcpcOptionSet()

---

## dhcpcOptionSet()

**NAME** dhcpcOptionSet() – add an option to the option request list

**SYNOPSIS**

```

STATUS dhcpcOptionSet
(
    void * pCookie,          /* identifier returned by dhcpcInit() */
    int   option            /* RFC 2132 tag of desired option */
)

```

**DESCRIPTION** This routine specifies which options the lease indicated by the *pCookie* parameter will request from a server. The *option* parameter specifies an option tag as defined in RFC 2132. See the **dhcpc/dhcp.h** include file for a listing of defined aliases for the available option

tags. This routine will not accept the following *option* values, which are either used by the server for control purposes or only supplied by the client:

```
_DHCP_PAD_TAG  
_DHCP_REQUEST_IPADDR_TAG  
_DHCP_LEASE_TIME_TAG  
_DHCP_OPT_OVERLOAD_TAG  
_DHCP_MSGTYPE_TAG  
_DHCP_SERVER_ID_TAG  
_DHCP_REQ_LIST_TAG  
_DHCP_ERRMSG_TAG  
_DHCP_MAXMSGSIZE_TAG  
_DHCP_CLASS_ID_TAG  
_DHCP_CLIENT_ID_TAG  
_DHCP_END_TAG
```

This routine also will not accept *option* values 62 or 63, which are not currently defined.

The maximum length of the option field in a DHCP message depends on the MTU size of the associated interface and the maximum DHCP message size set during the DHCP library initialization. Both the option request list and the options sent by the client through the **dhcpcOptionAdd()** routine share that field. Options which exceed the limit will not be stored.

---

**NOTE:** The boot program automatically requests all options necessary for default target configuration. This routine is only necessary to support special circumstances in which additional options are required. Any options requested in that case may be retrieved after the runtime image has started.

---

**NOTE:** The DHCP specification forbids changing the option request list after a lease has been established. Therefore, this routine must not be used after the **dhcpcBind()** call (in a runtime image) or the **dhcpcBootBind()** call (for a boot image). Changing the request list at that point could have unpredictable results.

---

**NOTE:** Options are added directly to outgoing DHCP messages, and numeric options (*e.g.*, lease duration time) are expected to be provided in network byte order. Care must be taken on little-endian hosts to insure that numeric arguments are properly byte-swapped before being passed to this routine.

---

**RETURNS** OK if the option was set successfully, or **ERROR** if the option is invalid or storage failed.

**ERRNO** S\_dhcpcLib\_BAD\_OPTION, S\_dhcpcLib\_OPTION\_NOT\_STORED

**SEE ALSO** dhcpcCommonLib

---

## dhcpcParamsGet()

**NAME** `dhcpcParamsGet()` – retrieve current configuration parameters

**SYNOPSIS**

```
STATUS dhcpcParamsGet
(
    void *          pCookie, /* identifier returned by dhcpcInit() */
    struct dhcp_param * pParamList /* requested parameters */
)
```

**DESCRIPTION**

This routine copies the current configuration parameters for the lease specified by the *pCookie* argument to the user-supplied and allocated `dhcp_param` structure referenced in *pParamList*. Within this structure, defined in `h/dhcp/dhcpc.h`, you should supply buffer pointers for the parameters that interest you. Set all other structure members to zero. When `dhcpcParamsGet()` returns, the buffers you specified in the submitted `dhcp_param` structure will contain the information you requested. This assumes that the specified lease is in the bound state and that DHCP knows that the lease parameters are good.

---

**NOTE:** The `temp_sname` and `temp_file` members of the `dhcp_param` structure are for internal use only. They reference temporary buffers for options that are passed using the `sname` and `file` members. Do not request either `temp_sname` or `temp_file`. Instead, request either `sname` or `file` if you want those parameters.

---

Many of the parameters within the user-supplied structure use one of the following secondary data types: `struct in_addr`, `struct u_shorts`, and `struct vendor_list`. Each of those structures accepts a length designation and a data pointer. For the first two data types, the `num` member indicates the size of the buffer in terms of the number of underlying elements. For example, the `STATIC_ROUTE` option returns one or more IP address pairs. Thus, setting the `num` member to 2 in the `static_route` entry would indicate that the corresponding buffer contained 16 bytes. By contrast, the `len` member in the `struct vendor_list` data type consists of the buffer size, in bytes. See RFC 1533 for specific details on the types of data for each option.

On return, each of the length designators are set to indicate the amount of data returned. For instance, the `num` member in the `static_route` entry could be set to 1 to indicate that only one IP address pair of 8 bytes was available.

**RETURNS** OK if in bound state, or `ERROR` otherwise.

**ERRNO** `S_dhcpcLib_BAD_COOKIE`, `S_dhcpcLib_NOT_INITIALIZED`, `S_dhcpcLib_NOT_BOUND`

**SEE ALSO** `dhcpcLib`

---

## dhcpcParamsShow()

<b>NAME</b>	<b>dhcpcParamsShow()</b> – display current lease parameters
<b>SYNOPSIS</b>	<pre>STATUS dhcpcParamsShow (     void * pCookie          /* identifier returned by dhcpcInit() */ )</pre>
<b>DESCRIPTION</b>	This routine prints all lease parameters for the lease identified by <i>pCookie</i> . It has no effect if the indicated lease is not currently active.
<b>RETURNS</b>	OK, or ERROR if lease identifier unknown.
<b>ERRNO</b>	S_dhcpcLib_BAD_COOKIE
<b>SEE ALSO</b>	dhcpcShow

---

## dhcpcRelease()

<b>NAME</b>	<b>dhcpcRelease()</b> – relinquish specified lease
<b>SYNOPSIS</b>	<pre>STATUS dhcpcRelease (     void * pCookie          /* identifier returned by dhcpcInit() */ )</pre>
<b>DESCRIPTION</b>	<p>This routine schedules the lease identified by the <i>pCookie</i> parameter for immediate release, regardless of time remaining, and removes all the associated data structures. After the release completes, a new call to <b>dhcpcInit()</b> is required before attempting another lease.</p> <hr/> <p><b>NOTE:</b> This routine will disable the underlying network interface if automatic configuration was requested. This may occur without warning if no event hook is installed.</p> <hr/>
<b>RETURNS</b>	OK if release scheduled, or ERROR otherwise.
<b>ERRNO</b>	S_dhcpcLib_BAD_COOKIE, S_dhcpcLib_NOT_INITIALIZED
<b>SEE ALSO</b>	dhcpcLib

---

## dhcpcServerGet()

<b>NAME</b>	<code>dhcpcServerGet()</code> – retrieve the current DHCP server
<b>SYNOPSIS</b>	<pre>STATUS dhcpcServerGet (     void *          pCookie,    /* identifier returned by dhcpcInit() */     struct in_addr * pServerAddr /* location for address of server */ )</pre>
<b>DESCRIPTION</b>	This routine returns the DHCP server that supplied the configuration parameters for the lease specified by the <i>pCookie</i> argument. This information is available only if the lease is in the bound state.
<b>RETURNS</b>	OK if in bound state and server available, or <b>ERROR</b> otherwise.
<b>ERRNO</b>	S_dhcpcLib_BAD_COOKIE, S_dhcpcLib_NOT_INITIALIZED, S_dhcpcLib_NOT_BOUND
<b>SEE ALSO</b>	<code>dhcpcLib</code>

---

## dhcpcServerShow()

<b>NAME</b>	<code>dhcpcServerShow()</code> – display current DHCP server
<b>SYNOPSIS</b>	<pre>STATUS dhcpcServerShow (     void * pCookie           /* identifier returned by dhcpcInit() */ )</pre>
<b>DESCRIPTION</b>	This routine prints the IP address of the DHCP server that provided the parameters for the lease identified by <i>pCookie</i> . It has no effect if the indicated lease is not currently active.
<b>RETURNS</b>	OK, or <b>ERROR</b> if lease identifier unknown.
<b>ERRNO</b>	S_dhcpcLib_BAD_COOKIE
<b>SEE ALSO</b>	<code>dhcpcShow</code>

## **dhcpcShowInit()**

<b>NAME</b>	<b>dhcpcShowInit()</b> – initialize the DHCP show facility
<b>SYNOPSIS</b>	<code>void dhcpcShowInit (void)</code>
<b>DESCRIPTION</b>	This routine links the DHCP show facility into the VxWorks system image. It is called from <code>usrNetwork.c</code> automatically if <code>INCLUDE_DHCP</code> and <code>INCLUDE_NET_SHOW</code> are defined at the time the image is constructed.
<b>SEE ALSO</b>	<code>dhcpcShow</code>

---

## **dhcpcShutdown()**

<b>NAME</b>	<b>dhcpcShutdown()</b> – disable DHCP client library
<b>SYNOPSIS</b>	<code>STATUS dhcpcShutdown (void)</code>
<b>DESCRIPTION</b>	This routine schedules the lease monitor task to clean up memory and exit, after releasing all currently active leases. The network boot device will be disabled if the DHCP client was used to obtain the VxWorks boot parameters and the resulting lease is still active. Any other interfaces using the addressing information from leases set for automatic configuration will also be disabled. Notification of a disabled interface will not occur unless an event hook has been installed. After the processing started by this request completes, the DHCP client library is unavailable until restarted with the <code>dhcpcLibInit()</code> routine.
<b>RETURNS</b>	<code>OK</code> if shutdown scheduled, or <code>ERROR</code> otherwise.
<b>ERRNO</b>	<code>S_dhcpcLib_NOT_INITIALIZED</code>
<b>SEE ALSO</b>	<code>dhcpcLib</code>

---

## dhcpcTimerGet()

**NAME** `dhcpcTimerGet()` – retrieve current lease timers

**SYNOPSIS**

```
STATUS dhcpcTimerGet
(
    void * pCookie,          /* identifier returned by dhcpcInit() */
    int * pT1,              /* time until lease renewal */
    int * pT2               /* time until lease rebinding */
)
```

**DESCRIPTION** This routine returns the number of clock ticks remaining on the timers governing the DHCP lease specified by the *pCookie* argument. This information is only available if the lease is in the bound state. Therefore, this routine will return **ERROR** if a BOOTP reply was accepted.

**RETURNS** OK if in bound state and values available, or **ERROR** otherwise.

**ERRNO** `S_dhcpcLib_BAD_COOKIE`, `S_dhcpcLib_NOT_INITIALIZED`, `S_dhcpcLib_NOT_BOUNDED`, `S_dhcpcLib_OPTION_NOT_PRESENT`, `S_dhcpcLib_TIMER_ERROR`

**SEE ALSO** `dhcpcLib`

---

## dhcpcTimersShow()

**NAME** `dhcpcTimersShow()` – display current lease timers

**SYNOPSIS**

```
STATUS dhcpcTimersShow
(
    void * pCookie          /* identifier returned by dhcpcInit() */
)
```

**DESCRIPTION** This routine prints the time remaining with each of the DHCP lease timers for the lease identified by *pCookie*. It has no effect if the indicated lease is not currently active.

**RETURNS** OK if show routine completes, or **ERROR** otherwise.

**ERRNO** `S_dhcpcLib_BAD_COOKIE`

**SEE ALSO** `dhcpcShow`

---

## dhcpcVerify()

**NAME** dhcpcVerify() – renew an established lease

**SYNOPSIS**

```
STATUS dhcpcVerify
(
    void * pCookie          /* identifier returned by dhcpcInit() */
)
```

**DESCRIPTION** This routine schedules the lease identified by the *pCookie* parameter for immediate renewal according to the process described in RFC 1541. If the renewal is unsuccessful, the lease negotiation process restarts. The routine is valid as long as the lease is currently active. The routine is also called automatically in response to a **dhcpcBind()** call for an existing lease.

---

**NOTE:** This routine is only intended for active leases obtained with the **dhcpcBind()** routine. It should not be used for parameters resulting from the **dhcpcInformGet()** routine.

---

**NOTE:** This routine will disable the underlying network interface if the verification fails and automatic configuration was requested. This may occur without warning if no event hook is installed.

---

**RETURNS** OK if verification scheduled, or **ERROR** otherwise.

**ERRNO** S\_dhcpcLib\_BAD\_COOKIE, S\_dhcpcLib\_NOT\_INITIALIZED, S\_dhcpcLib\_NOT\_BOUND

**SEE ALSO** dhcpcLib

---

## dhcpsAddressHookAdd()

**NAME** dhcpsAddressHookAdd() – assign a permanent address storage hook for the server

**SYNOPSIS**

```
STATUS dhcpsAddressHookAdd
(
    FUNCPTR pCacheHookRtn  /* routine to store/retrieve lease entries */
)
```

**DESCRIPTION** This routine allows the server to access some form of permanent storage to preserve additional address entries across restarts. This routine is not required, but leases using

unsaved addresses are not renewed. The only argument provided is the name of a function with the following interface:

```
STATUS dhcpsAddressStorageHook (int op,  
                                char *name, char *start, char *end,  
                                char *params);
```

The first parameter of this storage routine specifies one of the following operations:

```
DHCPS_STORAGE_START  
DHCPS_STORAGE_READ  
DHCPS_STORAGE_WRITE  
DHCPS_STORAGE_STOP
```

In response to a **START**, the storage routine should prepare to return data or overwrite data provided by earlier **WRITE** operations. For a **WRITE**, the storage routine must save the contents of the four buffers to permanent storage. Those buffers contain the **NULL**-terminated strings received by the **dhcpsLeaseEntryAdd()** routine. For a **READ**, the storage routine should copy previously stored data (as **NULL**-terminated strings) into the provided buffers in the order received by earlier **WRITE** operations. For a **STOP**, the storage routine should do any necessary cleanup. After a **STOP**, the storage routine should return an **ERROR** for all operations except **START**. However, the **STOP** operation does not normally occur since the server only deliberately exits following an unrecoverable error. This storage routine must not rely on that operation to handle **READ**, **WRITE**, or new **START** attempts.

The storage routine should return **OK** if successful, **ERROR** otherwise.

Note that, unlike the lease storage routine, there is no **CLEAR** operation.

Before the server is initialized, VxWorks calls this routine automatically passing in the function named in **DHCPS\_ADDRESS\_HOOK**.

<b>RETURNS</b>	<b>OK</b> , or <b>ERROR</b> if function pointer is <b>NULL</b> .
<b>ERRNO</b>	N/A
<b>SEE ALSO</b>	<b>dhcpsLib</b>

## dhcpsInit()

**NAME**            **dhcpsInit()** – set up the DHCP server parameters and data structures

**SYNOPSIS**        **STATUS** dhcpsInit  
                  (  
                  **DHCPS\_CFG\_PARAMS \* pDhcpCfg /\* configuration parameters \*/**  
                  )

**DESCRIPTION**    This routine creates the necessary data structures, builds the server address pool, retrieves any lease or address information from permanent storage through the user-provided hooks, and initializes the network interfaces for monitoring. It is called at system startup if **INCLUDE\_DHCP**S is defined at the time the VxWorks image is built.

The *maxSize* parameter specifies the maximum length supported for any DHCP message, including the UDP and IP headers and the largest link level header for all supported devices. The smallest valid value is 576 bytes, corresponding to the minimum IP datagram a host must accept. Larger values will allow the server to handle longer DHCP messages.

**RETURNS**        **OK**, or **ERROR** if could not initialize.

**SEE ALSO**        **dhcpsLib**

---

## dhcpsLeaseEntryAdd()

**NAME**            **dhcpsLeaseEntryAdd()** – add another entry to the address pool

**SYNOPSIS**        **STATUS** dhcpsLeaseEntryAdd  
                  (  
                  **char \* pName,                    /\* name of lease entry \*/**  
                  **char \* pStartIp,                /\* first IP address to assign \*/**  
                  **char \* pEndIp,                    /\* last IP address in assignment range \*/**  
                  **char \* pParams                /\* formatted string of lease parameters \*/**  
                  )

**DESCRIPTION**    This routine allows the user to add new entries to the address pool without rebuilding the VxWorks image. The routine requires a unique entry name of up to eight characters, starting and ending IP addresses, and a colon-separated list of parameters. Possible values for the parameters are listed in the reference entry for **dhcpsLib**. The parameters also determine the type of lease, which the server uses to determine priority when assigning lease addresses. For examples of possible lease types, see the reference entry for **dhcpsLib**.

**RETURNS** OK if entry read successfully, or **ERROR** otherwise.

**ERRNO** N/A

**SEE ALSO** **dhcpsLib**

---

## **dhcpsLeaseHookAdd()**

**NAME** **dhcpsLeaseHookAdd()** – assign a permanent lease storage hook for the server

**SYNOPSIS**

```
STATUS dhcpsLeaseHookAdd  
(  
    FUNCPTR pCacheHookRtn    /* routine to store/retrieve lease records */  
)
```

**DESCRIPTION** This routine allows the server to access some form of permanent storage that it can use to store current lease information across restarts. The only argument to **dhcpsLeaseHookAdd()** is a pointer to a storage routine with the following interface:

```
STATUS dhcpsStorageHook (int op, char *buffer, int datalen);
```

The first parameter of the storage routine specifies one of the following operations:

```
DHCPS_STORAGE_START  
DHCPS_STORAGE_READ  
DHCPS_STORAGE_WRITE  
DHCPS_STORAGE_STOP  
DHCPS_STORAGE_CLEAR
```

In response to **START**, the storage routine should prepare to return data or overwrite data provided by earlier **WRITES**. For a **WRITE**, the storage routine must save the contents of the buffer to permanent storage. For a **READ**, it should copy data previously stored into the provided buffer as a **NULL**-terminated string in FIFO order. For a **CLEAR**, the storage routine should discard currently stored data. After a **CLEAR**, the **READ** operation must return **ERROR** until additional data is stored. For a **STOP**, the storage routine must handle cleanup. After a **STOP**, **READ** and **WRITE** operations must return error until a **START** is received. Each of these operations must return **OK** if successful, or **ERROR** otherwise.

Before the server is initialized, VxWorks automatically calls **dhcpsLeaseHookAdd()**, passing in the routine name defined by **DHCPS\_LEASE\_HOOK**.

**RETURNS** **OK**, or **ERROR** if routine is **NULL**.

**ERRNO** N/A

**SEE ALSO** **dhcpsLib**

**difftime()**

---

**difftime()**

**NAME** difftime() – compute the difference between two calendar times (ANSI)

**SYNOPSIS**

```
double difftime
(
    time_t time1,          /* later time, in seconds */
    time_t time0          /* earlier time, in seconds */
)
```

**DESCRIPTION** This routine computes the difference between two calendar times: *time1* - *time0*.

**INCLUDE FILES** time.h

**RETURNS** The time difference in seconds, expressed as a double.

**SEE ALSO** ansiTime

---

**dirList()**

**NAME** dirList() – list contents of a directory (multi-purpose)

**SYNOPSIS**

```
STATUS dirList
(
    int    fd,              /* file descriptor to write on */
    char * dirName,        /* name of the directory to be listed */
    BOOL   doLong,         /* if TRUE, do long listing */
    BOOL   doTree           /* if TRUE, recurse into subdirs */
)
```

**DESCRIPTION** This command is similar to UNIX ls. It lists the contents of a directory in one of two formats. If *doLong* is FALSE, only the names of the files (or subdirectories) in the specified directory are displayed. If *doLong* is TRUE, then the file name, size, date, and time are displayed. If *doTree* flag is TRUE, then each subdirectory encountered will be listed as well (*i.e.*, the listing will be recursive).

The *dirName* parameter specifies the directory to be listed. If *dirName* is omitted or NULL, the current working directory will be listed. *dirName* may contain wildcard characters to list some of the directory's contents.

**LIMITATIONS**

- With **dosFsLib** file systems, MS-DOS volume label entries are not reported.
- Although an output format very similar to UNIX "ls" is employed, some information items have no particular meaning on some file systems.
- Some file systems which do not support the POSIX compliant **dirLib()** interface, can not support the *doLong* and *doTree* options.

**RETURNS** OK or ERROR.**SEE ALSO** **usrFsLib**, **dirLib**, **dosFsLib**, **ls()**, **ll()**, **lsr()**, **llr()**

---

## diskFormat()

**NAME** **diskFormat()** – format a disk**SYNOPSIS**  

```
STATUS diskFormat  
(  
    const char * pDevName    /* name of the device to initialize */  
)
```

**DESCRIPTION** This command formats a disk and creates a file system on it. The device must already have been created by the device driver and initialized for use with a particular file system, via **dosFsDevInit()**.This command calls **ioctl()** to perform the **FIODISKFORMAT** function.**EXAMPLE**

```
-> diskFormat "/fd0/"
```

**RETURNS** OK, or ERROR if the device cannot be opened or formatted.**SEE ALSO** **usrFsLib**, **dosFsLib**, *VxWorks Programmer's Guide: Target Shell*

**diskInit()**

---

**diskInit()**

<b>NAME</b>	<b>diskInit()</b> – initialize a file system on a block device
<b>SYNOPSIS</b>	<pre> <b>STATUS</b> diskInit (     const char * pDevName    /* name of the device to initialize */ ) </pre>
<b>DESCRIPTION</b>	<p>This function is now obsolete, use of <b>dosFsVolFormat()</b> is recommended.</p> <p>This command creates a new, blank file system on a block device. The device must already have been created by the device driver and initialized for use with a particular file system, via <b>dosFsDevCreate()</b>.</p>
<b>EXAMPLE</b>	<pre> -&gt; <b>diskInit</b> "/fd0/" </pre> <p>Note that if the disk is unformatted, it can not be mounted, thus <b>open()</b> will return error, in which case use the <b>dosFsVolFormat()</b> routine manually.</p> <p>This routine performs the <b>FIODISKINIT</b> ioctl operation.</p>
<b>RETURNS</b>	OK, or <b>ERROR</b> if the device cannot be opened or initialized.
<b>SEE ALSO</b>	<b>usrFsLib</b> , <b>dosFsLib</b> , <i>VxWorks Programmer's Guide: Target Shell</i>

---

**distCtl()**

<b>NAME</b>	<b>distCtl()</b> – perform a distributed objects control function (VxFusion Opt.)
<b>SYNOPSIS</b>	<pre> <b>int</b> distCtl (     <b>int</b> function,          /* function code */     <b>int</b> argument          /* arbitrary argument */ ) </pre>
<b>DESCRIPTION</b>	<p>This routine sets various parameters and hooks that control the system. It uses a syntax similar to that of the <b>ioctl()</b> routine. It accepts the following functions:</p> <p><b>DIST_CTL_LOG_HOOK</b></p> <p>This function sets a routine to be called each time a log message is produced. By default, the log hook writes the message to standard output. The prototype of the</p>

`log()` routine should look like this:

```
void log (char *logMsg);
```

#### **DIST\_CTL\_PANIC\_HOOK**

This function sets a routine to be called when the system panics. By default, the panic hook writes the panic message to standard output. The `panic()` routine must not return. The prototype of the `panic()` routine should look like this:

```
void panic (char *panicMsg);
```

#### **DIST\_CTL\_RETRY\_TIMEOUT**

This function sets the initial send retry timeout in clock ticks. If no ACK is received within a timeout period, the packet is resent. The default value and granularity of `DIST_CTL_RETRY_TIMEOUT` is system dependent.

<b>vxWorks Version</b>	<b>Default Value</b>	<b>Granularity</b>
5.4 and below	1000ms	500ms
5.5 and AE	200ms	100ms

`DIST_CTL_RETRY_TIMEOUT` is designated in ticks, but rounded down to a multiple of the system's granularity. The timeout period for the  $n$ th send is:

$$n * \text{DIST\_CTL\_RETRY\_TIMEOUT}$$

#### **DIST\_CTL\_MAX\_RETRIES**

This function sets a limit for the number of retries when sending fails. The default value is system dependent, but is set to 5 for all current versions of vxWorks.

#### **DIST\_CTL\_NACK\_SUPPORT**

This function enables or disables the sending of negative acknowledgments (NACKs). NACKs are used to request a resend of a single missing fragment from a packet. They are sent immediately after a fragment is found to be missing. If `arg` is `FALSE` (0), the sending of negative acknowledgments is disabled. If `arg` is `TRUE` (1), the sending of NACKs is enabled. By default, NACKs are enabled.

#### **DIST\_CTL\_PGGYBAK\_UNICAST\_SUPPORT**

This function enables or disables unicast piggy-backing. When unicast piggy-backing is enabled, the system waits some time until it sends an acknowledgment for a previously received packet. In the meantime, if a data packet is sent to a host already awaiting an acknowledgment, the acknowledgment is delivered (that is, piggy-backed) with the data packet. Enabling piggy-backing is useful for reducing the number of packets sent; however, it increases latency if no data packets are sent while the system waits. When unicast piggy-backing is disabled, an acknowledgment is delivered immediately in its own packet. This function turns piggy-backing on and off for unicast communication only. If `arg` is `FALSE` (0), unicast piggy-backing is disabled. If `arg` is `TRUE` (1), unicast piggy-backing is enabled. By default, piggy-backing is disabled for unicast communication.

**distCtl()****DIST\_CTL\_PGGYBAK\_BRDCST\_SUPPORT**

This function enables or disables broadcast piggy-backing. When broadcast piggy-backing is enabled, the system waits some time until it sends an acknowledgment for a previously received packet. In the meantime, if a data packet is sent to a host already awaiting an acknowledgment, the acknowledgment is delivered (that is, piggy-backed) with the data packet. Enabling piggy-backing is useful for reducing the number of packets sent; however, it increases latency if no broadcast data packets are sent while the system waits. When broadcast piggy-backing is disabled, an acknowledgment is delivered immediately in its own packet. This function turns piggy-backing on and off for broadcast communication only. If *arg* is **FALSE** (0), broadcast piggy-backing is disabled. If *arg* is **TRUE** (1), broadcast piggy-backing is enabled. By default, piggy-backing is disabled for broadcast communication.

**DIST\_CTL\_OPERATIONAL\_HOOK**

This function adds a routine to a list of routines to be called each time a node shifts to the operational state. A maximum of 8 routines can be added to the list. The prototype of each **operational()** routine should look as follows:

```
void operational (DIST_NODE_ID nodeStateChanged);
```

**DIST\_CTL\_CRASHED\_HOOK**

This function adds a routine to a list of routines to be called each time a node shifts to the crashed state. A node shifts to the crashed state when it does not acknowledge a message within the maximum number of retries. The list can contain a maximum of 8 routines; however VxFusion supplies one routine, leaving room for only 7 user-supplied routines. The prototype of each **crashed()** routine should look as follows:

```
void crashed (DIST_NODE_ID nodeStateChanged);
```

**DIST\_CTL\_GET\_LOCAL\_ID**

This function returns the local node ID.

**DIST\_CTL\_GET\_LOCAL\_STATE**

This function returns the state of the local node.

**DIST\_CTL\_SERVICE\_HOOK**

This function sets a routine to be called each time a service fails, for a service invoked by a remote node. The *argument* parameter is a pointer to a **servError()** routine. The prototype of the **servError()** routine should look as follows:

```
void servError (int servId, int status);
```

The system is aware of the following services:

```
DIST_ID_MSG_Q_SERV      (0) /* message queue service      */
DIST_ID_MSG_Q_GRP_SERV (1) /* message queue group service */
DIST_ID_DNDB_SERV      (2) /* distributed name database    */
DIST_ID_DGDB_SERV      (3) /* distributed group database    */
DIST_ID_INCO_SERV      (4) /* incorporation protocol       */
DIST_ID_GAP_SERV       (5) /* group agreement protocol     */
```

**DIST\_CTL\_SERVICE\_CONF**

This function configures a specified service. The *argument* parameter is a pointer to a **DIST\_SERV\_CONF** structure which holds the service ID and its configuration to be set. **DIST\_SERV\_CONF** is defined as follows:

```
typedef struct
{
    int servId;      /* ID of service to configure */
    int taskPrio;   /* priority of service task */
    int netPrio;    /* network priority of service */
} DIST_SERV_CONF;
```

The system is aware of the following services:

```
DIST_ID_MSG_Q_SERV      (0) /* message queue service */
DIST_ID_MSG_Q_GRP_SERV (1) /* message queue group service */
DIST_ID_DNDB_SERV      (2) /* distributed name database */
DIST_ID_DGDB_SERV      (3) /* distributed group database */
DIST_ID_INCO_SERV      (4) /* incorporation protocol */
DIST_ID_GAP_SERV       (5) /* group agreement protocol */
```

If one of the configuration parameters is -1, it remains unchanged. The parameter *taskPrio* can range from 0 to 255; *netPrio* can range from 0 to 7.

A service's configuration can be changed at any time.

<b>AVAILABILITY</b>	This routine is distributed as a component of the unbundled distributed message queues option, VxFusion.
<b>RETURNS</b>	OK or the value requested if <i>function</i> is known; <b>ERROR</b> if <i>function</i> is unknown or the argument is invalid.
<b>ERRNO</b>	<b>S_distLib_UNKNOWN_REQUEST</b> The control function is unknown.
<b>SEE ALSO</b>	<b>distLib</b>

---

## distIfShow()

**NAME** `distIfShow()` – display information about the installed interface adapter (VxFusion Opt.)

**SYNOPSIS** `STATUS distIfShow (void)`

**DESCRIPTION** This routine displays information about the installed interface adapter. It displays the configuration parameters, as well as some statistical data.

**EXAMPLE**

```
-> distIfShow
Interface Name           : "UDP adapter"
MTU                      : 1500
Network Header Size     : 14
SWP Buffer                : 32
Maximum Number of Fragments : 10
Maximum Length of Packet : 14860
Broadcast Address       : 0x930b26ff
Telegrams received      : 23
Telegrams received for sending : 62
Incoming Telegrams discarded : 0
Outgoing Telegrams discarded : 0
```

In this example, the installed interface adapter has the name “UDP adapter.” The largest telegram that can be transmitted without fragmentation is 1500 bytes long. The network header requires fourteen (14) of those bytes; therefore the largest amount of user data that can be transmitted without fragmentation is 1486 bytes. The sliding window protocol’s buffer has 32 entries, which results in a window of size 16. The number of fragments that the packet can be broken into is limited by the size of the sequence field in the network header. The example interface adapter can handle up to 10 fragments, which results in a maximum packet length of 14860 ((1500 - 14) \* 128) bytes. The broadcast address of this driver is 0x930b26ff (147.11.38.255). The last four lines of output show statistical data.

**AVAILABILITY** This routine is distributed as a component of the unbundled distributed message queues option, VxFusion.

**RETURNS** OK, or ERROR if there is no interface installed.

**SEE ALSO** `distIfShow`

---

## distInit()

**NAME** `distInit()` – initialize and bootstrap the current node (VxFusion Opt.)

**SYNOPSIS**

```

STATUS distInit
(
    DIST_NODE_ID myNodeId,      /* node ID of this node */
    FUNCPTR      ifInitRtn,    /* interface adapter init routine */
    void *       pIfInitConf,  /* ptr to interface configuration */
    int          maxTBuFsLog2, /* max number of telegram buffers */
    int          maxNodesLog2, /* max number of nodes in node db */
    int          maxQueuesLog2, /* max number of queues on this node */
    int          maxGroupsLog2, /* max number of groups in db */
    int          maxNamesLog2, /* max bindings in name db */
    int          waitNTicks    /* wait n ticks when bootstrapping */
)

```

**DESCRIPTION** This routine initializes VxFusion on the current node. The routine begins by initializing the local databases and other internal services. As part of this process, the current node is given the address specified by the *myNodeId* argument.

Secondly, this routine links a network driver to the stack by calling the interface adapter initialization routine specified by the *ifInitRtn* argument. If the interface adapter initialization is successful, this routine then initializes the telegram buffer library which is needed for manipulating telegram buffers—the buffers that hold the packets sent between nodes.

Thirdly, this routine attempts to determine what other VxFusion nodes are active on the network. This is done by continually sending a **BOOTSTRAP** telegram, which indicates to other nodes that VxFusion is starting up on this node. Nodes that receive a **BOOTSTRAP** telegram answer by sending an **XACK** telegram. The **XACK** telegram contains information about the remote node. The sender of the first **XACK** received is the godfather for the current node. The purpose of the godfather is to update local databases. If no **XACK** is received within the amount of time specified by the *waitNTicks* argument, it is assumed that this node is the first node to come up on the network.

As soon as a godfather is located or it is assumed that a node sending an **XACK** is the first to do so on the network, the state of the node shifts from the *booting* state to the *network* state. In the network state, all packets are sent using reliable communication channels; therefore all packets must be now acknowledged by the receiver(s).

If a godfather has been located, the current node asks it to update the local databases by sending an **INCO\_REQ** packet. The godfather then begins updating the local databases. When the godfather finishes the update, it sends an **INCO\_DONE** packet to the node being updated.

**distInit()**

Once the database updates have completed, the node moves into the *operational* state and broadcasts an INCO\_UPNOW packet.

The number of telegram buffers pre-allocated is equal to  $2^{\text{maxTBuFsLog2}}$ .

Up to  $2^{\text{maxNodesLog2}}$  nodes can be handled by the node database.

The number of distributed message queues is limited to  $2^{\text{maxQueuesLog2}}$ .

Distributed message queue groups may not exceed  $2^{\text{maxGroupsLog2}}$  groups.

The distributed name database can work with up to  $2^{\text{maxNamesLog2}}$  entries.

**EXAMPLE**

```
-> distInit (0x930b2610, distIfUdpInit, "ln0", 9, 5, 7, 6, 8,
(4*sysClkRateGet()))
```

This command sets the ID of the local node to 0x930b2610 (147.11.38.16). The **distIfUdpInit()** routine is called to initialize the interface adapter (in this case, a UDP adapter). The UDP adapter requires a pointer to the hardware interface name as configuration data (in this case, "ln0"). When starting up, 512 ( $2^9$ ) telegram buffers are pre-allocated. The node database is configured to hold as many as 32 ( $2^5$ ) nodes, including the current node. 128 ( $2^7$ ) distributed message queues can be created on the local node. The local group database can hold up to 64 ( $2^6$ ) groups, while the name database is limited to 256 ( $2^8$ ) entries.

When the node bootstraps, it waits for 4 seconds ( $4 * \text{sysClkRateGet}()$ ) to allow other nodes to respond.

---

**NOTE:** This routine is called automatically with default parameters when a target boots using a VxWorks image with VxFusion installed.

---

**AVAILABILITY**

This routine is distributed as a component of the unbundled distributed message queues option, VxFusion.

**RETURNS**

OK, or ERROR if the initialization fails.

**SEE ALSO**

**distLib**, **distLib**

## distNameAdd()

**NAME** `distNameAdd()` – add an entry to the distributed name database (VxFusion Opt.)

**SYNOPSIS**

```

STATUS distNameAdd
(
    char *      name,      /* name to enter in database */
    void *      value,    /* ptr to value to associate with name */
    int         valueLen, /* size of value in bytes */
    DIST_NAME_TYPE type   /* type associated with name */
)

```

**DESCRIPTION** This routine adds the name of a specified object, along with its type and value, to the distributed objects distributed name database. All copies of the distributed name database within the system are updated.

The *name* parameter is an arbitrary, null-terminated string with a maximum of 20 characters, including the null terminator.

The value associated with *name* is located at *value* and is of length *valueLen*, currently limited to 8 bytes.

By convention, *type* values of less than 0x1000 are reserved by VxWorks; all other values are user definable. The following types are pre-defined in **distNameLib.h**:

Type Name	Value	Datum
<code>_DIST_MSG_Q</code>	=0	distributed message queue
<code>_DIST_NODE</code>	= 16	node ID
<code>_DIST_UINT8</code>	= 64	8-bit unsigned integer
<code>_DIST_UINT16</code>	= 65	16-bit unsigned integer
<code>_DIST_UINT32</code>	= 66	32-bit unsigned integer
<code>T_DIST_UINT64</code>	= 67	64-bit unsigned integer
<code>T_DIST_FLOAT</code>	= 68	float (32-bit)
<code>T_DIST_DOUBLE</code>	= 69	double (64-bit)

The byte-order of pre-defined types is preserved in a byte-order-heterogeneous network.

The value (and type!) bound to a symbolic name can be changed by calling **distNameAdd()** with a new value (and type).

This routine returns **OK**, even if some nodes on the system do not respond to the add request broadcast. A node that does not acknowledge a transmission is assumed to have crashed. You can use the **distCtl()** routine in **distLib** to set a routine to be called in the event that a node crashes.

---

**NOTE:** If you add a distributed object ID (`T_DIST_MSG_Q`) to the database, another reference to the object is built. This reference is stored in the database. After the return of `distNameAdd()`, *value* holds the reference (a new object ID). Use the ID returned by `distNameAdd()` each time you want to address the global object bound to *name*. Subsequent updates of the binding in the database are transparent. The original object ID specifies exactly the locally created object.

---

<b>AVAILABILITY</b>	This routine is distributed as a component of the unbundled distributed message queues option, VxFusion.
<b>RETURNS</b>	OK, or ERROR if the operation fails.
<b>ERRNO</b>	<code>S_distNameLib_NAME_TOO_LONG</code> The name being added to the database is too long. <code>S_distNameLib_ILLEGAL_LENGTH</code> The argument <i>valueLen</i> is not in the range 1 to 8. <code>S_distNameLib_DATABASE_FULL</code> The database is full. <code>S_distNameLib_INCORRECT_LENGTH</code> The argument <i>valueLen</i> is incorrect for the pre-defined <i>type</i> .
<b>SEE ALSO</b>	<code>distNameLib</code> , <code>distLib</code>

---

## distNameFilterShow()

<b>NAME</b>	<code>distNameFilterShow()</code> – display the distributed name database filtered by type (VxFusion Opt.)
<b>SYNOPSIS</b>	<pre>void distNameFilterShow (     DIST_NAME_TYPE type      /* type to filter the database by */ )</pre>
<b>DESCRIPTION</b>	This routine displays the contents of the distributed name database filtered by <i>type</i> . The data displayed includes the symbolic ASCII name, the <i>type</i> , and the value. If the <i>type</i> is not pre-defined, it is printed in decimal and the value shown in a hex dump.

---

**NOTE:** Option `VX_FP_TASK` should be set when spawning any task in which `distNameFilterShow()` is called unless it is certain that no floating point values will be displayed. The target shell has this option set.

---

```

EXAMPLE      -> distNameFilterShow(0)
                NAME                TYPE                VALUE
                -----
dmq-01          T_DIST_MSG_Q  0x3ff9fb
dmq-02          T_DIST_MSG_Q  0x3ff98b
dmq-03          T_DIST_MSG_Q  0x3ff94b
dmq-04          T_DIST_MSG_Q  0x3ff8db
dmq-05          T_DIST_MSG_Q  0x3ff89b
grp1            T_DIST_MSG_Q  0x3ff9bb
grp2            T_DIST_MSG_Q  0x3ff90b
value = 0 = 0x0

```

**AVAILABILITY** This routine is distributed as a component of the unbundled distributed message queues option, VxFusion.

**RETURNS** N/A

**SEE ALSO** **distNameShow**

---

## distNameFind()

**NAME** **distNameFind()** – find an object by name in the local database (VxFusion Opt.)

**SYNOPSIS**

```

STATUS distNameFind
(
    char *          name,      /* name to search for */
    void * *       pValue,    /* where to return ptr to value */
    DIST_NAME_TYPE * pType,   /* where to return type */
    int            waitType /* NO_WAIT or WAIT_FOREVER */
)

```

**DESCRIPTION** This routine searches the distributed name database for an object matching a specified *name*. If the object is found, a pointer to the value and its type are copied to the address pointed to by *pValue* and *pType*. If the type is **T\_DIST\_MSG\_Q**, the identifier returned can be used with generic message queue handling routines in **msgQLib**, such as **msgQSend()**, **msgQReceive()**, and **msgQNumMsgs()**.

**AVAILABILITY** This routine is distributed as a component of the unbundled distributed message queues option, VxFusion.

**RETURNS** **OK**, or **ERROR** if the search fails.

<b>ERRNO</b>	<b>S_distNameLib_NAME_TOO_LONG</b> The name to be found in the database is too long.
	<b>S_distNameLib_INVALID_WAIT_TYPE</b> The wait type should be either <b>NO_WAIT</b> or <b>WAIT_FOREVER</b> .
<b>SEE ALSO</b>	<b>distNameLib</b>

---

## distNameFindByValueAndType()

**NAME** **distNameFindByValueAndType()** – look up the name of an object by value and type  
(VxFusion Opt.)

**SYNOPSIS**

```
STATUS distNameFindByValueAndType  
(  
    void *      value,      /* value to search for */  
    DIST_NAME_TYPE type,    /* type of object for which to search */  
    char *      name,      /* where to return name */  
    int         waitType   /* NO_WAIT or WAIT_FOREVER */  
)
```

**DESCRIPTION** This routine searches the distributed name database for an object matching a specified *value* and *type*. If the object is found, its name is copied to the address pointed to by *name*.

---

**NOTE:** Unlike the **smNameFindByValue()** routine, used with the shared-memory objects name database, this routine must know the type of the object being searched for. Searching on the value only might not return a unique object.

---

**AVAILABILITY** This routine is distributed as a component of the unbundled distributed message queues option, VxFusion.

**RETURNS** **OK**, or **ERROR** if the search fails.

**ERRNO** **S\_distNameLib\_INVALID\_WAIT\_TYPE**  
The wait type should be either **NO\_WAIT** or **WAIT\_FOREVER**.

**SEE ALSO** **distNameLib**

---

## distNameRemove()

<b>NAME</b>	<b>distNameRemove()</b> – remove an entry from the distributed name database (VxFusion Opt.)
<b>SYNOPSIS</b>	<pre><b>STATUS distNameRemove</b> <b>(</b>     <b>char * name</b>                <b>/* name of object to remove */</b> <b>)</b></pre>
<b>DESCRIPTION</b>	<p>This routine removes an object, that is bound to <i>name</i>, from the distributed name database. All copies of the distributed name database get updated.</p> <p>This routine returns <b>OK</b>, even if some nodes on the system do not respond to the remove request broadcast. A node that does not acknowledge a transmission is assumed to have crashed. You can use the <b>distCtl()</b> routine in <b>distLib</b> to set a routine to be called in the event that a node crashes.</p> <p>Removing the name of a distributed object ID (<b>T_DIST_MSG_Q</b>) does not invalidate the object ID.</p>
<b>AVAILABILITY</b>	This routine is distributed as a component of the unbundled distributed message queues option, VxFusion.
<b>URNS</b>	<b>OK</b> , or <b>ERROR</b> if the operation fails.
<b>ERRNO</b>	<b>S_distNameLib_NAME_TOO_LONG</b> The name to be removed from the database is too long.
<b>SEE ALSO</b>	<b>distNameLib</b> , <b>distLib</b>

---

## distNameShow()

<b>NAME</b>	<b>distNameShow()</b> – display the entire distributed name database (VxFusion Opt.)
<b>SYNOPSIS</b>	<pre><b>void distNameShow (void)</b></pre>
<b>DESCRIPTION</b>	This routine displays the entire contents of the distributed name database. The data displayed includes the symbolic ASCII name, the type, and the value. If the type is not pre-defined, it is printed in decimal and the value shown in a hex dump.

---

**NOTE:** Option `VX_FP_TASK` should be set when spawning any task in which `distNameShow()` is called unless it is certain that no floating point values will be in the database. The target shell has this option set.

---

**EXAMPLE**

```
-> distNameShow()
      NAME                TYPE                VALUE
-----
nile                T_DIST_NODE  0x930b2617  (2466981399)
columbia            T_DIST_NODE  0x930b2616  (2466981398)
dmq-01              T_DIST_MSG_Q 0x3ff9fb
dmq-02              T_DIST_MSG_Q 0x3ff98b
dmq-03              T_DIST_MSG_Q 0x3ff94b
dmq-04              T_DIST_MSG_Q 0x3ff8db
dmq-05              T_DIST_MSG_Q 0x3ff89b
gData                4096 0x48 0x65 0x6c 0x6c 0x6f 0x00
gCount              T_DIST_UINT32 0x2d (45)
grp1                 T_DIST_MSG_Q 0x3ff9bb
grp2                 T_DIST_MSG_Q 0x3ff90b
value = 0 = 0x0
```

**AVAILABILITY**

This routine is distributed as a component of the unbundled distributed message queues option, VxFusion.

**RETURNS**

N/A

**SEE ALSO**

`distNameShow`

---

## distTBufAlloc()

**NAME**

`distTBufAlloc()` – allocate a telegram buffer from the pool of buffers (VxFusion Opt.)

**SYNOPSIS**

```
DIST_TBUF * distTBufAlloc (void)
```

**DESCRIPTION**

This routine allocates a telegram buffer from a pre-allocated pool of telegram buffers. It is the responsibility of the caller to use the `distTBufFree()` routine to free the buffer when the caller is finished with it.

**AVAILABILITY**

This routine is distributed as a component of the unbundled distributed message queues option, VxFusion.

**RETURNS**

A pointer to a `DIST_TBUF`, or `NULL` if the allocation fails.

**SEE ALSO**

`distTBufLib`, `distTBufFree()`

---

## distTBufFree()

<b>NAME</b>	<b>distTBufFree()</b> – return a telegram buffer to the pool of buffers (VxFusion Opt.)
<b>SYNOPSIS</b>	<pre>void distTBufFree (     DIST_TBUF * pTBuf          /* ptr to buffer to be returned to pool */ )</pre>
<b>DESCRIPTION</b>	This routine returns a buffer previously allocated to a caller back to the pool of free telegram buffers.
<b>AVAILABILITY</b>	This routine is distributed as a component of the unbundled distributed message queues option, VxFusion.
<b>RETURNS</b>	N/A
<b>SEE ALSO</b>	<b>distTBufLib, distTBufAlloc()</b>

---

## div()

<b>NAME</b>	<b>div()</b> – compute a quotient and remainder (ANSI)
<b>SYNOPSIS</b>	<pre>div_t div (     int numer,                /* numerator */     int denom                 /* denominator */ )</pre>
<b>DESCRIPTION</b>	<p>This routine computes the quotient and remainder of <i>numer/denom</i>. If the division is inexact, the resulting quotient is the integer of lesser magnitude that is the nearest to the algebraic quotient. If the result cannot be represented, the behavior is undefined; otherwise, <b>quot</b> * <i>denom</i> + <b>rem</b> equals <i>numer</i>.</p> <p>This routine is not reentrant. For a reentrant version, see <b>div_r()</b>.</p>
<b>INCLUDE FILES</b>	<b>stdlib.h</b>
<b>RETURNS</b>	A structure of type <b>div_t</b> , containing both the quotient and the remainder.
<b>SEE ALSO</b>	<b>ansiStdlib</b>

**div\_r()**


---

<b>NAME</b>	<b>div_r()</b> – compute a quotient and remainder (reentrant)
<b>SYNOPSIS</b>	<pre> void div_r (     int    numer,          /* numerator */     int    denom,         /* denominator */     div_t * divStructPtr  /* div_t structure */ ) </pre>
<b>DESCRIPTION</b>	<p>This routine computes the quotient and remainder of <i>numer</i> / <i>denom</i>. The quotient and remainder are stored in the <b>div_t</b> structure pointed to by <i>divStructPtr</i>.</p> <p>This routine is the reentrant version of <b>div()</b>.</p>
<b>INCLUDE FILES</b>	<b>stdlib.h</b>
<b>RETURNS</b>	N/A
<b>SEE ALSO</b>	<b>ansiStdlib</b>

---

**dosFsChkDsk()**

<b>NAME</b>	<b>dosFsChkDsk()</b> – make volume integrity checking.
<b>SYNOPSIS</b>	<pre> STATUS dosFsChkDsk (     DOS_FILE_DESC_ID pFd,    /* file descriptor of root dir */     u_int            params  /* check level and verbosity */ ) </pre>
<b>DESCRIPTION</b>	<p>This library does not makes integrity check process itself, but instead uses routine provided by <b>dosChkLib</b>. This routine prepares parameters and invokes checking routine via a pre-initialized function pointer. If <b>dosChkLib</b> does not configured into vxWorks, this routine returns <b>ERROR</b>.</p> <p>Ownership on device should be taken by an upper level routine.</p>
<b>RETURNS</b>	<b>STATUS</b> as returned by volume checking routine or <b>ERROR</b> , if such routine does not installed.
<b>ERRNO</b>	<b>S_dosFsLib_UNSUPPORTED</b> .
<b>SEE ALSO</b>	<b>dosFsLib</b>

## dosFsDevCreate()

**NAME** dosFsDevCreate() – create file system device.

**SYNOPSIS**

```

STATUS dosFsDevCreate
(
    char *          pDevName,      /* device name */
    CBIO_DEV_ID    cbio,          /* CBIO or cast blkIo device */
    u_int          maxFiles,      /* max no. of simultaneously open files */
    u_int          autoChkLevel /* automate volume integrity check level */
                                /* via mounting 0 - default: DOS_CHK_REPAIR
                                DOS_CHK_VERB_1 */
)

```

**DESCRIPTION** This routine associates a CBIO device with a logical I/O device name and prepare it to perform file system functions. It takes a `CBIO_DEV_ID` device handle, typically created by `dcacheDevCreate()`, and defines it as a dosFs volume. As a result, when high-level I/O operations (e.g., `open()`, `write()`) are performed on the device, the calls will be routed through `dosFsLib`. The `pCbio` parameter is the handle of the underlying cache or block device.

The argument `maxFiles` specifies the number of files that can be opened at once on the device.

The volume structure integrity can be automatically checked during volume mounting. Parameter `autoChkLevel` defines checking level (`DOS_CHK_ONLY` or `DOS_CHK_REPAIR`), that can be bitwise or-ed with check verbosity level value (`DOS_CHK_VERB_SILENT`, `DOS_CHK_VERB_1` or `DOS_CHK_VERB_2`). If value of `autoChkLevel` is 0, this means default level, that is `DOS_CHK_REPAIR | DOS_CHK_VERB_1`. To prevent check disk autocall, set `autoChkLevel` to `NONE`.

Note that actual disk accesses are deferred to the time when `open()` or `creat()` are first called. That is also when the automatic disk checking will take place. Therefore this function will succeed in cases where a removable disk is not present in the drive.

**RETURNS** OK, or `ERROR` if the device name is already in use or insufficient memory.

**SEE ALSO** `dosFsLib`

---

## dosFsLastAccessDateEnable()

**NAME** `dosFsLastAccessDateEnable()` – enable last access date updating for this volume

**SYNOPSIS**

```
STATUS dosFsLastAccessDateEnable
(
    DOS_VOLUME_DESC_ID dosVolDescId, /* dosfs volume ID to alter */
    BOOL                enable      /* TRUE = enable update, FALSE = */
                                /* disable update */
)
```

**DESCRIPTION** This function enables or disables updating of the last access date directory entry field on open-read-close operations for the given dosFs volume. The last access date file indicates the last date that a file has been read or written. When the optional last access date field update is enabled, read operations on a file will cause a write to the media.

**RETURNS** OK or ERROR if the volume is invalid or enable is not TRUE or FALSE.

**SEE ALSO** `dosFsLib`

---

## dosFsLibInit()

**NAME** `dosFsLibInit()` – prepare to use the dosFs library

**SYNOPSIS**

```
STATUS dosFsLibInit
(
    int ignored
)
```

**DESCRIPTION** This routine initializes the dosFs library. This routine installs `dosFsLib` as a driver in the I/O system driver table, and allocates and sets up the necessary structures. The driver number assigned to `dosFsLib` is placed in the global variable `dosFsDrvNum`.

**RETURNS** OK or ERROR, if driver can not be installed.

**SEE ALSO** `dosFsLib`

---

## dosFsShow()

**NAME** `dosFsShow()` – display dosFs volume configuration data.

**SYNOPSIS**

```
STATUS dosFsShow
(
    void * pDevName,          /* name of device */
    u_int level              /* detail level */
)
```

**DESCRIPTION** This routine obtains the dosFs volume configuration for the named device, formats the data, and displays it on the standard output.

If no device name is specified, the current default device is described.

**RETURNS** OK or ERROR, if no valid device specified.

**SEE ALSO** `dosFsLib`

---

## dosFsVolDescGet()

**NAME** `dosFsVolDescGet()` – convert a device name into a DOS volume descriptor pointer.

**SYNOPSIS**

```
DOS_VOLUME_DESC_ID dosFsVolDescGet
(
    void * pDevNameOrPVolDesc, /* device name or pointer to dos vol desc */
    u_char * * ppTail          /* return ptr for name, used in iosDevFind */
)
```

**DESCRIPTION** This routine validates *pDevNameOrPVolDesc* to be a DOS volume descriptor pointer else a path to a DOS device. This routine uses the standard `iosLib` function `iosDevFind()` to obtain a pointer to the device descriptor. If device is eligible, *ppTail* is filled with the pointer to the first character following the device name. Note that *ppTail* is passed to `iosDevFind()`. *ppTail* may be passed as NULL, in which case it is ignored.

**RETURNS** A `DOS_VOLUME_DESC_ID` or NULL if not a DOSFS device.

**ERRNO** `S_dosFsLib_INVALID_PARAMETER`

**SEE ALSO** `dosFsLib`

---

## dosFsVolFormat( )

**NAME** dosFsVolFormat( ) – format an MS-DOS compatible volume

**SYNOPSIS**

```
STATUS dosFsVolFormat
(
    void * device,          /* device name or volume or CBIO pointer */
    int    opt,            /* bit-wise or'ed options */
    FUNCPTR pPromptFunc    /* interactive parameter change callback */
)
```

**DESCRIPTION** This utility routine performs the initialization of file system data structures on a disk. It supports FAT12 for small disks, FAT16 for medium size and FAT32 for large volumes. The *device* argument may be either a device name known to the I/O system, or a **dosFsLib** Volume descriptor or a CBIO device handle.

The *opt* argument is a bit-wise or'ed combination of options controlling the operation of this routine as follows:

**DOS\_OPT\_DEFAULT**

If the current volume boot block is reasonably intact, use existing parameters, else calculate parameters based only on disk size, possibly reusing only the volume label and serial number.

**DOS\_OPT\_PRESERVE**

Attempt to preserve the current volume parameters even if they seem to be somewhat unreliable.

**DOS\_OPT\_BLANK**

Disregard the current volume parameters, and calculate new parameters based only on disk size.

**DOS\_OPT\_QUIET**

Do not produce any diagnostic output during formatting.

**DOS\_OPT\_FAT16**

Format the volume with FAT16 format even if the disk is larger than 2 Gbytes, which would normally be formatted with FAT32.

**DOS\_OPT\_FAT32**

Format the volume with FAT32, even if the disk is smaller than 2 Gbytes, but is larger than 512 Mbytes.

**DOS\_OPT\_VXLONGNAMES**

Format the volume to use Wind River proprietary case-sensitive Long File Names. Note that this format is incompatible with any other implementation of the MS-DOS file system.

The third argument, *pPromptFunc* is an optional pointer to a function that may interactively prompt the user to change any of the modifiable volume parameters before formatting:

```
void formatPromptFunc( DOS_VOL_CONFIG *pConfig );
```

The `<*pConfig>` structure upon entry to **formatPromptFunc()** will contain the initial volume parameters, some of which can be changed before it returns. *pPromptFunc* should be NULL if no interactive prompting is required.

<b>COMPATIBILITY</b>	Although this routine tries to format the disk to be compatible with Microsoft implementations of the FAT and FAT32 file systems, there may be differences which are not under WRS control. For this reason, it is highly recommended that any disks which are expected to be interchanged between vxWorks and Windows should be formatted under Windows to provide the best interchangeability. The WRS implementation is more flexible, and should be able to handle the differences when formatting is done on Windows, but Windows implementations may not be able to handle minor differences between their implementation and ours.
<b>AVAILABILITY</b>	This function is an optional part of the MS-DOS file system, and may be included in a target system if it is required to be able to format new volumes.
<b>RETURNS</b>	OK or ERROR if was unable to format the disk.
<b>SEE ALSO</b>	<b>dosFsFmtLib</b>

---

## dosSetVolCaseSens()

<b>NAME</b>	<b>dosSetVolCaseSens()</b> – set case sensitivity of volume
<b>SYNOPSIS</b>	<pre><b>STATUS dosSetVolCaseSens</b> <b>(</b>     <b>DOS_VOLUME_DESC_ID pVolDesc,</b>     <b>BOOL sensitivity</b> <b>)</b></pre>
<b>DESCRIPTION</b>	Pass TRUE to setup a case sensitive volume. Pass FALSE to setup a case insensitive volume. Note this affects rename lookups only.
<b>RETURNS</b>	TRUE if <i>pVolDesc</i> pointed to a DOS volume.
<b>SEE ALSO</b>	<b>dosFsLib</b>

---

## dpartDevCreate()

**NAME** dpartDevCreate() – initialize a partitioned disk

**SYNOPSIS**

```
CBIO_DEV_ID dpartDevCreate
(
    CBIO_DEV_ID subDev,          /* lower level CBIO device */
    int          nPart,          /* # of partitions */
    FUNCPTR     pPartDecodeFunc /* function to decode partition table */
)
```

**DESCRIPTION**

To handle a partitioned disk, this function should be called, with *subDev* as the handle returned from **dcacheDevCreate()**, It is recommended that for efficient operation a single disk cache be allocated for the entire disk and shared by its partitions.

*nPart* is the maximum number of partitions which are expected for the particular disk drive. Up to 24 (C-Z) partitions per disk are supported.

### PARTITION DECODE FUNCTION

An external partition table decode function is provided via the *pPartDecodeFunc* argument, which implements a particular style and format of partition tables, and fill in the results into a table defined as Pn array of **PART\_TABLE\_ENTRY** types. See **dpartCbio.h** for definition of **PART\_TABLE\_ENTRY**. The prototype for this function is as follows:

```
STATUS parDecodeFunc
(
    CBIO_DEV_ID dev,          /* device from which to read blocks */
    PART_TABLE_ENTRY *pPartTab, /* table where to fill results */
    int nPart                /* # of entries in <pPartTable> */
)
```

**RETURNS** CBIO\_DEV\_ID or NULL if error creating CBIO device.

**SEE ALSO** dpartCbio, dosFsDevCreate().

---

## dpartPartGet()

**NAME** `dpartPartGet()` – retrieve handle for a partition

**SYNOPSIS**

```
CBIO_DEV_ID dpartPartGet
(
    CBIO_DEV_ID masterHandle, /* CBIO handle of the master partition */
    int         partNum      /* partition number from 0 to nPart */
)
```

**DESCRIPTION** This function retrieves a CBIO handle into a particular partition of a partitioned device. This handle is intended to be used with `dosFsDevCreate()`.

**RETURNS** `CBIO_DEV_ID` or `NULL` if partition is out of range, or *masterHandle* is invalid.

**SEE ALSO** `dpartCbio`, `dosFsDevCreate()`

---

## dspInit()

**NAME** `dspInit()` – initialize DSP support

**SYNOPSIS**

```
void dspInit (void)
```

**DESCRIPTION** This routine initializes DSP support and must be called before using the DSP. This is done automatically by the root task, `usrRoot()`, in `usrConfig.c` when `INCLUDE_DSPis` defined in `configAll.h`.

**RETURNS** N/A

**SEE ALSO** `dspLib`

## **dspShowInit()**

**NAME** `dspShowInit()` – initialize the DSP show facility

**SYNOPSIS** `void dspShowInit (void)`

**DESCRIPTION** This routine links the DSP show facility into the VxWorks system. The facility is included automatically when `INCLUDE_SHOW_ROUTINES` and `INCLUDE_DSP` are defined in `configAll.h`.

**RETURNS** N/A

**SEE ALSO** `dspShow`

---

## **dspTaskRegsShow()**

**NAME** `dspTaskRegsShow()` – print the contents of a task's DSP registers

**SYNOPSIS** `void dspTaskRegsShow`  
`(`  
`int task                          /* task to display dsp registers for */`  
`)`

**DESCRIPTION** This routine prints to standard output the contents of a task's DSP registers.

**RETURNS** N/A

**SEE ALSO** `dspShow`

---

## e()

**NAME** e() – set or display eventpoints (WindView)

**SYNOPSIS**

```

STATUS e
(
    INSTR * addr,           /* where to set eventpoint, or 0 means */
                           /* display all eventpoints */
    event_t eventId,       /* event ID */
    int     taskNameOrId,  /* task affected; 0 means all tasks */
    FUNCPTR evtRtn,       /* function to be invoked; NULL means no */
                           /* function is invoked */
    int     arg            /* argument to be passed to evtRtn */
)

```

**DESCRIPTION**

This routine sets “eventpoints”--that is, breakpoint-like instrumentation markers that can be inserted in code to generate and log an event for use with WindView. Event logging must be enabled with **wvEvtLogEnable()** for the eventpoint to be logged.

*eventId* selects the eventpoint number that will be logged: it is in the user event ID range (0-25536).

If *addr* is **NULL**, then all eventpoints and breakpoints are displayed. If *taskNameOrId* is 0, then this event is logged in all tasks. The *evtRtn* routine is called when this eventpoint is hit. If *evtRtn* returns **OK**, then the eventpoint is logged; otherwise, it is ignored. If *evtRtn* is a **NULL** pointer, then the eventpoint is always logged.

Eventpoints are exactly like breakpoints (which are set with the **b()** command) except in how the system responds when the eventpoint is hit. An eventpoint typically records an event and continues immediately (if *evtRtn* is supplied, this behavior may be different). Eventpoints cannot be used at interrupt level.

To delete an eventpoint, use **bd()**.

**RETURNS**

**OK**, or **ERROR** if *addr* is odd or nonexistent in memory, or if the breakpoint table is full.

**SEE ALSO**

**dbgLib**, **wvEvent()**

**edi()**

---

**edi()**

<b>NAME</b>	<b>edi()</b> – return the contents of register <b>edi</b> (also <b>esi</b> - <b>eax</b> ) (x86/SimNT)
<b>SYNOPSIS</b>	<pre>int edi (     int taskId           /* task ID, 0 means default task */ )</pre>
<b>DESCRIPTION</b>	<p>This command extracts the contents of register <b>edi</b> from the TCB of a specified task. If <i>taskId</i> is omitted or zero, the last task referenced is assumed.</p> <p>Similar routines are provided for all address registers (<b>edi</b> - <b>eax</b>): <b>edi()</b> - <b>eax()</b>.</p> <p>The stack pointer is accessed via <b>eax()</b>.</p>
<b>RETURNS</b>	The contents of register <b>edi</b> (or the requested register).
<b>SEE ALSO</b>	<b>dbgArchLib</b> , <i>VxWorks Programmer's Guide: Debugging</i>

---

**eflags()**

<b>NAME</b>	<b>eflags()</b> – return the contents of the status register (x86/SimNT)
<b>SYNOPSIS</b>	<pre>int eflags (     int taskId           /* task ID, 0 means default task */ )</pre>
<b>DESCRIPTION</b>	<p>This command extracts the contents of the status register from the TCB of a specified task. If <i>taskId</i> is omitted or zero, the last task referenced is assumed.</p>
<b>RETURNS</b>	The contents of the status register.
<b>SEE ALSO</b>	<b>dbgArchLib</b> , <i>VxWorks Programmer's Guide: Debugging</i>

---

## endFindByName()

**NAME** `endFindByName()` – find a device using its string name

**SYNOPSIS**

```
END_OBJ * endFindByName
(
    char * pName,          /* device name to search for */
    int   unit
)
```

**DESCRIPTION** This routine takes a string name and a unit number and finds the device that has that name/unit combination.

### VXWORKS AE PROTECTION DOMAINS

Under VxWorks AE, you can call `endFindByName()` from within the kernel protection domain only, and the data referenced in the *pName* parameter must reside in the kernel protection domain. In addition, the returned `END_OBJ` is valid in the kernel protection domain only. This restriction does not apply under non-AE versions of VxWorks.

**RETURNS** A pointer to an `END_OBJ`; or `NULL`, if the device is not found.

**SEE ALSO** `muxLib`

---

## envLibInit()

**NAME** `envLibInit()` – initialize environment variable facility

**SYNOPSIS**

```
STATUS envLibInit
(
    BOOL installHooks
)
```

**DESCRIPTION** If *installHooks* is `TRUE`, task create and delete hooks are installed that will optionally create and destroy private environments for the task being created or destroyed, depending on the state of `VX_PRIVATE_ENV` in the task options word. If *installHooks* is `FALSE` and a task requires a private environment, it is the application's responsibility to create and destroy the private environment, using `envPrivateCreate()` and `envPrivateDestroy()`.

**RETURNS** `OK`, or `ERROR` if an environment cannot be allocated or the hooks cannot be installed.

**SEE ALSO** `envLib`

## envPrivateCreate()

**NAME** envPrivateCreate() – create a private environment

**SYNOPSIS**

```
STATUS envPrivateCreate
(
    int taskId,          /* task to have private environment */
    int envSource       /* -1 = make an empty private environment 0 */
                        /* = copy global env to new private env taskId */
                        /* = copy the specified task's env */
)
```

**DESCRIPTION** This routine creates a private set of environment variables for a specified task, if the environment variable task create hook is not installed.

**RETURNS** OK, or ERROR if memory is insufficient.

**SEE ALSO** envLib, envLibInit(), envPrivateDestroy()

---

## envPrivateDestroy()

**NAME** envPrivateDestroy() – destroy a private environment

**SYNOPSIS**

```
STATUS envPrivateDestroy
(
    int taskId          /* task with private env to destroy */
)
```

**DESCRIPTION** This routine destroys a private set of environment variables that were created with **envPrivateCreate()**. Calling this routine is unnecessary if the environment variable task create hook is installed and the task was spawned with **VX\_PRIVATE\_ENV**.

**RETURNS** OK, or ERROR if the task does not exist.

**SEE ALSO** envLib, envPrivateCreate()

---

## envShow()

**NAME** `envShow()` – display the environment for a task

**SYNOPSIS**

```
void envShow
(
    int taskId          /* task for which environment is printed */
)
```

**DESCRIPTION** This routine prints to standard output all the environment variables for a specified task. If *taskId* is `NULL`, then the calling task's environment is displayed.

**RETURNS** N/A

**SEE ALSO** `envLib`

---

## errnoGet()

**NAME** `errnoGet()` – get the error status value of the calling task

**SYNOPSIS** `int errnoGet (void)`

**DESCRIPTION** This routine gets the error status stored in `errno`. It is provided for compatibility with previous versions of VxWorks and simply accesses `errno` directly.

**RETURNS** The error status value contained in `errno`.

**SEE ALSO** `errnoLib`, `errnoSet()`, `errnoOfTaskGet()`

## errnoOfTaskGet()

- NAME** `errnoOfTaskGet()` – get the error status value of a specified task
- SYNOPSIS**
- ```
int errnoOfTaskGet
(
    int taskId           /* task ID, 0 means current task */
)
```
- DESCRIPTION**
- This routine gets the error status most recently set for a specified task. If *taskId* is zero, the calling task is assumed, and the value currently in **errno** is returned.
- This routine is provided primarily for debugging purposes. Normally, tasks access **errno** directly to set and get their own error status values.
- RETURNS** The error status of the specified task, or **ERROR** if the task does not exist.
- SEE ALSO** `errnoLib`, `errnoSet()`, `errnoGet()`
- 

## errnoOfTaskSet()

- NAME** `errnoOfTaskSet()` – set the error status value of a specified task
- SYNOPSIS**
- ```
STATUS errnoOfTaskSet
(
    int taskId,           /* task ID, 0 means current task */
    int errorValue       /* error status value */
)
```
- DESCRIPTION**
- This routine sets the error status for a specified task. If *taskId* is zero, the calling task is assumed, and **errno** is set with the specified error status.
- This routine is provided primarily for debugging purposes. Normally, tasks access **errno** directly to set and get their own error status values.
- RETURNS** **OK**, or **ERROR** if the task does not exist.
- SEE ALSO** `errnoLib`, `errnoSet()`, `errnoOfTaskGet()`

---

## errnoSet()

**NAME** `errnoSet()` – set the error status value of the calling task

**SYNOPSIS**

```
STATUS errnoSet
(
    int errorValue          /* error status value to set */
)
```

**DESCRIPTION** This routine sets the `errno` variable with a specified error status. It is provided for compatibility with previous versions of VxWorks and simply accesses `errno` directly.

**RETURNS** OK, or ERROR if the interrupt nest level is too deep.

**SEE ALSO** `errnoLib`, `errnoGet()`, `errnoOfTaskSet()`

---

## etherMultiAdd()

**NAME** `etherMultiAdd()` – add multicast address to a multicast address list

**SYNOPSIS**

```
int etherMultiAdd
(
    LIST * pList,          /* pointer to list of multicast addresses */
    char* pAddress        /* address you want to add to list */
)
```

**DESCRIPTION** This routine adds an Ethernet multicast address list for a given END. The address is a six-byte value pointed to by *pAddress*.

**RETURNS** OK or ENETRESET.

**SEE ALSO** `etherMultiLib`

---

## etherMultiDel()

**NAME** etherMultiDel() – delete an Ethernet multicast address record

**SYNOPSIS**

```
int etherMultiDel
(
    LIST * pList,           /* pointer to list of multicast addresses */
    char* pAddress         /* address you want to add to list */
)
```

**DESCRIPTION** This routine deletes an Ethernet multicast address from the list. The address is a six-byte value pointed to by *pAddress*.

**RETURNS** OK or ENETRESET.

**SEE ALSO** etherMultiLib

---

## etherMultiGet()

**NAME** etherMultiGet() – retrieve a table of multicast addresses from a driver

**SYNOPSIS**

```
int etherMultiGet
(
    LIST*      pList,       /* pointer to list of multicast addresses */
    MULTI_TABLE* pTable    /* table into which to copy addresses */
)
```

**DESCRIPTION** This routine runs down the multicast address list stored in a driver and places all the entries it finds into the multicast table structure passed to it.

**RETURNS** OK or ERROR.

**SEE ALSO** etherMultiLib

---

## eventClear()

<b>NAME</b>	<b>eventClear()</b> – clear all events for current task.
<b>SYNOPSIS</b>	<b>STATUS eventClear (void)</b>
<b>DESCRIPTION</b>	This function clears all received events for the calling task.
<b>RETURNS</b>	OK on success or ERROR.
<b>ERRNO</b>	S_intLib_NOT_ISR_CALLABLE Routine has been called from interrupt level.
<b>SEE ALSO</b>	eventLib

---

## eventReceive()

<b>NAME</b>	<b>eventReceive()</b> – wait for event(s)
<b>SYNOPSIS</b>	<pre><b>STATUS eventReceive</b> (      <b>UINT32</b>  <b>events,</b>           /* events task is waiting to occur */      <b>UINT8</b>   <b>options,</b>         /* user options */      <b>int</b>     <b>timeout,</b>         /* ticks to wait */      <b>UINT32</b> * <b>pEventsReceived</b> /* events occurred are returned through this */  )</pre>
<b>DESCRIPTION</b>	<p>Pends task until one or all specified <i>events</i> have occurred. When they have, <i>pEventsReceived</i> will be filled with those that did occur.</p> <p>The <i>options</i> parameter is used for three user options. Firstly, it is used to specify if the task is going to wait for all events to occur or only one of them. One of the following has to be selected:</p> <p><b>EVENTS_WAIT_ANY</b> (0x1) only one event has to occur</p> <p><b>EVENTS_WAIT_ALL</b> (0x0) will wait until all events occur.</p> <p>Secondly, it is used to specify if the events returned in <i>pEventsReceived</i> will be only those received and wanted, or all events received (even the ones received before <b>eventReceive()</b>)</p>

**eventReceive()**

was called). By default it returns only the events wanted. Performing a bitwise-OR of the following:

**EVENTS\_RETURN\_ALL** (0x2)

causes the function to return received events, both wanted and unwanted.

Thirdly, it can be used to retrieve what events have been received by the current task. If the option

**EVENTS\_FETCH** (0x80)

is chosen by the user, then *pEventsReceived* will be filled with the events that have already been received and will return immediately. In this case, the parameters *events* and *timeout*, as well as all the other options, are ignored. Also, events are not cleared, allowing to get a peek at the events that have already been received.

The *timeout* parameter specifies the number of ticks to wait for wanted events to be sent to the waiting task. It can also have the following special values:

**NO\_WAIT** (0)

return immediately, even if no events have arrived.

**WAIT\_FOREVER** (-1)

never time out.

It must also be noted that events sent to the receiving task are cleared prior to returning, as if a call to **eventClear()** was done.

The parameter *pEventsReceived* is always filled with the events received even when the function returns an error, except if a value of **NULL** was passed.

---

**WARNING:** This routine may not be used from interrupt level.

---

**RETURNS**

OK on success or **ERROR**.

**ERRNO****S\_eventLib\_TIMEOUT**

Wanted events not received before specified time expired.

**S\_eventLib\_NOT\_ALL\_EVENTS**

Specified **NO\_WAIT** as the timeout parameter and wanted events were not already received when the routine was called.

**S\_objLib\_OBJ\_DELETED**

Task is waiting for some events from a resource that is subsequently deleted.

**S\_intLib\_NOT\_ISR\_CALLABLE**

Function has been called from ISR.

**S\_eventLib\_ZERO\_EVENTS**

The *events* parameter has been passed a value of 0.

**SEE ALSO**

**eventLib**, **semEvLib**, **msgQEvLib**, **eventSend()**

---

## eventSend()

<b>NAME</b>	<code>eventSend()</code> – send event(s)
<b>SYNOPSIS</b>	<pre> <b>STATUS</b> eventSend     (         <b>int</b>    taskId,           /* task events will be sent to */         <b>UINT32</b> events          /* events to send */     ) </pre>
<b>DESCRIPTION</b>	Sends specified event(s) to specified task. Passing a taskId of NULL sends events to the calling task.
<b>RETURNS</b>	OK on success or ERROR.
<b>ERRNO</b>	<p><code>S_objLib_OBJ_ID_ERROR</code> Task ID is invalid.</p> <p><code>S_eventLib_NULL_TASKID_AT_INT_LEVEL</code> Routine was called from ISR with a taskId of NULL.</p>
<b>SEE ALSO</b>	<code>eventLib</code> , <code>eventReceive()</code>

---

## excConnect()

<b>NAME</b>	<code>excConnect()</code> – connect a C routine to an exception vector (PowerPC)
<b>SYNOPSIS</b>	<pre> <b>STATUS</b> excConnect     (         <b>VOIDFUNCPTR</b> * vector,    /* exception vector to attach to */         <b>VOIDFUNCPTR</b>  routine    /* routine to be called */     ) </pre>
<b>DESCRIPTION</b>	<p>This routine connects a specified C routine to a specified exception vector. An exception stub is created and is placed at <i>vector</i> in the exception table. The address of <i>routine</i> is stored in the exception stub code. When an exception occurs, the processor jumps to the exception stub code, saves the registers, and calls the C routines.</p> <p>The routine can be any normal C code, except that it must not invoke certain operating system functions that may block or perform I/O operations.</p>

The registers are saved to an Exception Stack Frame (ESF) placed on the stack of the task that has produced the exception. The structure of the ESF used to save the registers is defined in `h/arch/ppc/esfPpc.h`.

The only argument passed by the exception stub to the C routine is a pointer to the ESF containing the registers values. The prototype of this C routine is described below:

```
void excHandler (ESFPPC *);
```

When the C routine returns, the exception stub restores the registers saved in the ESF and continues execution of the current task.

**RETURNS** OK, always.

**SEE ALSO** `excArchLib`, `excIntConnect()`, `excVecSet()`

---

## excCrtConnect()

**NAME** `excCrtConnect()` – connect a C routine to a critical exception vector (PowerPC 403)

**SYNOPSIS**

```
STATUS excCrtConnect  
(  
    VOIDFUNCPTR * vector,      /* exception vector to attach to */  
    VOIDFUNCPTR routine      /* routine to be called */  
)
```

**DESCRIPTION** This routine connects a specified C routine to a specified critical exception vector. An exception stub is created and in placed at *vector* in the exception table. The address of *routine* is stored in the exception stub code. When an exception occurs, the processor jumps to the exception stub code, saves the registers, and call the C routines.

The routine can be any normal C code, except that it must not invoke certain operating system functions that may block or perform I/O operations.

The registers are saved to an Exception Stack Frame (ESF) which is placed on the stack of the task that has produced the exception. The ESF structure is defined in `h/arch/ppc/esfPpc.h`.

The only argument passed by the exception stub to the C routine is a pointer to the ESF containing the register values. The prototype of this C routine is as follows:

```
void excHandler (ESFPPC *);
```

When the C routine returns, the exception stub restores the registers saved in the ESF and continues execution of the current task.

**RETURNS** OK, always.

**SEE ALSO** excArchLib, excIntConnect(), excIntCrtConnect(), excVecSet()

---

## excHookAdd()

**E**

**NAME** excHookAdd() – specify a routine to be called with exceptions

**SYNOPSIS**

```
void excHookAdd
(
    FUNCPTR excepHook      /* routine to call when exceptions occur */
)
```

**DESCRIPTION** This routine specifies a routine that will be called when hardware exceptions occur. The specified routine is called after normal exception handling, which includes displaying information about the error. Upon return from the specified routine, the task that incurred the error is suspended.

The exception handling routine should be declared as:

```
void myHandler
(
    int    task,      /* ID of offending task          */
    int    vecNum,   /* exception vector number       */
    ESFxx * pEsf     /* pointer to exception stack frame */
)
```

where *task* is the ID of the task that was running when the exception occurred. *ESFxx* is architecture-specific and can be found by examining */target/h/arch/arch/esfarch.h*; for example, the PowerPC uses ESFPPC.

This facility is normally used by **dbgLib()** to activate its exception handling mechanism. If an application provides its own exception handler, it will supersede the **dbgLib** mechanism.

**RETURNS** N/A

**SEE ALSO** excLib, excTask()

---

## exclnit()

<b>NAME</b>	<b>exclnit()</b> – initialize the exception handling package
<b>SYNOPSIS</b>	<b>STATUS</b> <b>exclnit</b> (void)
<b>DESCRIPTION</b>	This routine installs the exception handling facilities and spawns <b>excTask()</b> , which performs special exception handling functions that need to be done at task level. It also creates the message queue used to communicate with <b>excTask()</b> .
	<b>NOTE:</b> The exception handling facilities should be installed as early as possible during system initialization in the root task, <b>usrRoot()</b> , in <b>usrConfig.c</b> .
<b>RETURNS</b>	OK, or <b>ERROR</b> if a message queue cannot be created or <b>excTask()</b> cannot be spawned.
<b>SEE ALSO</b>	<b>excLib</b> , <b>excTask()</b>

---

## excIntConnect()

<b>NAME</b>	<b>excIntConnect()</b> – connect a C routine to an asynchronous exception vector (PowerPC, ARM)
<b>SYNOPSIS</b>	<pre><b>STATUS</b> <b>excIntConnect</b> (     <b>VOIDFUNCPTR</b> * <b>vector</b>,      /* exception vector to attach to */     <b>VOIDFUNCPTR</b>  <b>routine</b>     /* routine to be called */ )</pre>
<b>DESCRIPTION</b>	<p>This routine connects a specified C routine to a specified asynchronous exception vector. When the C routine is invoked, interrupts are still locked. It is the responsibility of the C routine to re-enable the interrupt.</p> <p>The routine can be any normal C code, except that it must not invoke certain operating system functions that may block or perform I/O operations.</p> <p><b>NOTE:</b> On PowerPC, the vector is typically the external interrupt vector 0x500 and the decremter vector 0x900. An interrupt stub is created and placed at <i>vector</i> in the exception table. The address of <i>routine</i> is stored in the interrupt stub code. When the asynchronous exception occurs the processor jumps to the interrupt stub code, saves only the requested registers, and calls the C routines. Before saving the requested registers, the</p>

interrupt stub switches from the current task stack to the interrupt stack. For nested interrupts, no stack-switching is performed, because the interrupt is already set.

**NOTE:** On the ARM, the address of *routine* is stored in a function pointer to be called by the stub installed on the IRQ exception vector following an asynchronous exception. This routine is responsible for determining the interrupt source and dispatching the correct handler for that source. Before calling the routine, the interrupt stub switches to SVC mode, changes to a separate interrupt stack and saves necessary registers. In the case of a nested interrupt, no SVC stack switch occurs.

**RETURNS** OK, always.

**SEE ALSO** excArchLib, excConnect(), excVecSet()

---

## excIntCrtConnect()

**NAME** excIntCrtConnect() – connect a C routine to a critical interrupt vector (PowerPC 403)

**SYNOPSIS**

```
STATUS excIntCrtConnect
(
    VOIDFUNCPTR * vector,      /* exception vector to attach to */
    VOIDFUNCPTR routine       /* routine to be called */
)
```

**DESCRIPTION** This routine connects a specified C routine to a specified asynchronous critical exception vector such as the critical external interrupt vector (0x100), or the watchdog timer vector (0x1020). An interrupt stub is created and placed at *vector* in the exception table. The address of *routine* is stored in the interrupt stub code. When the asynchronous exception occurs, the processor jumps to the interrupt stub code, saves only the requested registers, and calls the C routines.

When the C routine is invoked, interrupts are still locked. It is the C routine's responsibility to re-enable interrupts.

The routine can be any normal C routine, except that it must not invoke certain operating system functions that may block or perform I/O operations.

Before the requested registers are saved, the interrupt stub switches from the current task stack to the interrupt stack. In the case of nested interrupts, no stack switching is performed, because the interrupt stack is already set.

**RETURNS** OK, always.

**SEE ALSO** excArchLib, excConnect(), excCrtConnect(), excVecSet()

## excTask()

<b>NAME</b>	<b>excTask()</b> – handle task-level exceptions
<b>SYNOPSIS</b>	<pre>void excTask ()</pre>
<b>DESCRIPTION</b>	This routine is spawned as a task by <b>excInit()</b> to perform functions that cannot be performed at interrupt or trap level. It has a priority of 0. Do not suspend, delete, or change the priority of this task.
<b>RETURNS</b>	N/A
<b>SEE ALSO</b>	<b>excLib, excInit()</b>

---

## excVecGet()

<b>NAME</b>	<b>excVecGet()</b> – get a CPU exception vector (PowerPC, ARM)
<b>SYNOPSIS</b>	<pre>FUNCPTR excVecGet (     FUNCPTR * vector          /* vector offset */ )</pre>
<b>DESCRIPTION</b>	This routine returns the address of the C routine currently connected to <i>vector</i> .
<b>RETURNS</b>	The address of the C routine.
<b>SEE ALSO</b>	<b>excArchLib, excVecSet()</b>

---

## excVecInit()

<b>NAME</b>	<b>excVecInit()</b> – initialize the exception/interrupt vectors
<b>SYNOPSIS</b>	<pre>STATUS excVecInit (void)</pre>

**DESCRIPTION** This routine sets all exception vectors to point to the appropriate default exception handlers. These handlers will safely trap and report exceptions caused by program errors or unexpected hardware interrupts.

**MC680x0:**

All vectors from vector 2 (address 0x0008) to 255 (address 0x03fc) are initialized. Vectors 0 and 1 contain the reset stack pointer and program counter.

**x86:**

All vectors from vector 0 (address 0x0000) to 255 (address 0x07f8) are initialized to default handlers.

**MIPS:**

All MIPS exception, trap, and interrupt vectors are set to default handlers.

**x86:**

All vectors from vector 0 (address 0x0000) to 255 (address 0x07f8) are initialized to default handlers.

**PowerPC:**

There are 48 vectors and only vectors that are used are initialized.

**SH:**

There are 256 vectors, initialized with the default exception handler (for exceptions) or the uninitialized interrupt handler (for interrupts). On SH-2, vectors 0 and 1 contain the power-on reset program counter and stack pointer. Vectors 2 and 3 contain the manual reset program counter and stack pointer. On SH-3 and SH-4 processors the vector table is located at (vbr + 0x800), and the (exception code / 8) value is used as vector offset. The first two vectors are reserved for special use: "trapa #0" (offset 0x0) to implement software breakpoint, and "trapa #1" (offset 0x4) to detect integer zero divide exception.

**ARM:**

All exception vectors are initialized to default handlers except 0x14 (Address) which is now reserved on the ARM and 0x1C (FIQ), which is not used by VxWorks.

**SimSolaris/SimNT:**

This routine does nothing on both simulators and always returns **OK**.

---

**NOTE:** This routine is usually called from the system start-up routine, **usrInit()**, in **usrConfig.c**. It must be called before interrupts are enabled.

---

**RETURNS** OK, always.

**SEE ALSO** excArchLib, excLib

---

## excVecSet()

**NAME** excVecSet() – set a CPU exception vector (PowerPC, ARM)

**SYNOPSIS**

```
void excVecSet
(
    FUNCPTR * vector,          /* vector offset */
    FUNCPTR  function         /* address to place in vector */
)
```

**DESCRIPTION** This routine specifies the C routine that will be called when the exception corresponding to *vector* occurs. This routine does not create the exception stub; it simply replaces the C routine to be called in the exception stub.

---

**NOTE:** On the ARM, there is no **excConnect()** routine, unlike the PowerPC. The C routine is attached to a default stub using **excVecSet()**.

---

**RETURNS** N/A

**SEE ALSO** excArchLib, excVecGet(), excConnect(), excIntConnect()

---

## exit()

**NAME** exit() – exit a task (ANSI)

**SYNOPSIS**

```
void exit
(
    int code                  /* code stored in TCB for delete hooks */
)
```

**DESCRIPTION** This routine is called by a task to cease to exist as a task. It is called implicitly when the “main” routine of a spawned task is exited. The *code* parameter will be stored in the WIND\_TCB for possible use by the delete hooks, or post-mortem debugging.

**ERRNO** N/A

**SEE ALSO** taskLib, taskDelete(), *American National Standard for Information Systems -Programming Language - C, ANSI X3.159-1989: Input/Output (stdlib.h), VxWorks Programmer’s Guide: Basic OS*

---

## exp()

<b>NAME</b>	<code>exp()</code> – compute an exponential value (ANSI)
<b>SYNOPSIS</b>	<pre>double exp (     double x          /* exponent */ )</pre>
<b>DESCRIPTION</b>	This routine returns the exponential value of $x$ in double precision (IEEE double, 53 bits). A range error occurs if $x$ is too large.
<b>INCLUDE FILES</b>	<code>math.h</code>
<b>RETURNS</b>	The double-precision exponential value of $x$ . Special cases: If $x$ is +INF or NaN, <code>exp()</code> returns $x$ . If $x$ is -INF, it returns 0.
<b>SEE ALSO</b>	<code>ansiMath</code> , <code>mathALib</code>

---

## expf()

<b>NAME</b>	<code>expf()</code> – compute an exponential value (ANSI)
<b>SYNOPSIS</b>	<pre>float expf (     float x   /* exponent */ )</pre>
<b>DESCRIPTION</b>	This routine returns the exponential of $x$ in single precision.
<b>INCLUDE FILES</b>	<code>math.h</code>
<b>RETURNS</b>	The single-precision exponential value of $x$ .
<b>SEE ALSO</b>	<code>mathALib</code>

**fabs()**

---

<b>NAME</b>	<b>fabs()</b> – compute an absolute value (ANSI)
<b>SYNOPSIS</b>	<pre>double fabs (     double v    /* number to return the absolute value of */ )</pre>
<b>DESCRIPTION</b>	This routine returns the absolute value of <i>v</i> in double precision.
<b>INCLUDE FILES</b>	math.h
<b>RETURNS</b>	The double-precision absolute value of <i>v</i> .
<b>ERRNO</b>	EDOM, ERANGE
<b>SEE ALSO</b>	ansiMath, mathALib

---

**fabsf()**

<b>NAME</b>	<b>fabsf()</b> – compute an absolute value (ANSI)
<b>SYNOPSIS</b>	<pre>float fabsf (     float v    /* number to return the absolute value of */ )</pre>
<b>DESCRIPTION</b>	This routine returns the absolute value of <i>v</i> in single precision.
<b>INCLUDE FILES</b>	math.h
<b>RETURNS</b>	The single-precision absolute value of <i>v</i> .
<b>SEE ALSO</b>	mathALib

---

## fclose()

**NAME** `fclose()` – close a stream (ANSI)

**SYNOPSIS**

```
int fclose
(
    FILE * fp                /* stream to close */
)
```

**DESCRIPTION** This routine flushes a specified stream and closes the associated file. Any unwritten buffered data is delivered to the host environment to be written to the file; any unread buffered data is discarded. The stream is disassociated from the file. If the associated buffer was allocated automatically, it is deallocated.

**INCLUDE FILES** `stdio.h`

**RETURNS** Zero if the stream is closed successfully, or EOF if errors occur.

**ERRNO** EBADF

**SEE ALSO** `ansiStdio`, `fflush()`

---

## fdopen()

**NAME** `fdopen()` – open a file specified by a file descriptor (POSIX)

**SYNOPSIS**

```
FILE * fdopen
(
    int          fd,          /* file descriptor */
    const char * mode        /* mode to open with */
)
```

**DESCRIPTION** This routine opens the file specified by the file descriptor *fd* and associates a stream with it. The *mode* argument is used just as in the `fopen()` function.

**INCLUDE FILES** `stdio.h`

**RETURNS** A pointer to a stream, or a null pointer if an error occurs, with `errno` set to indicate the error.

**ERRNO** EINVAL

**SEE ALSO** `ansiStdio`, `fopen()`, `freopen()`, *Information Technology - POSIX - Part 1: System API [C Language]*, IEEE Std 1003.1

---

## fdprintf()

**NAME** `fdprintf()` – write a formatted string to a file descriptor

**SYNOPSIS**

```
int fdprintf
(
    int          fd,          /* file descriptor to write to */
    const char * fmt,        /* format string to write */
    ...          /* optional arguments to format */
)
```

**DESCRIPTION** This routine writes a formatted string to a specified file descriptor. Its function and syntax are otherwise identical to `printf()`.

**RETURNS** The number of characters output, or **ERROR** if there is an error during output.

**SEE ALSO** `fioLib`, `printf()`

---

## feof()

**NAME** `feof()` – test the end-of-file indicator for a stream (ANSI)

**SYNOPSIS**

```
int feof
(
    FILE * fp                /* stream to test */
)
```

**DESCRIPTION** This routine tests the end-of-file indicator for a specified stream.

**INCLUDE FILES** `stdio.h`

**RETURNS** Non-zero if the end-of-file indicator is set for *fp*.

**SEE ALSO** `ansiStdio`, `clearerr()`

---

## **ferror()**

**NAME** `ferror()` – test the error indicator for a file pointer (ANSI)

**SYNOPSIS**

```
int ferror
(
    FILE * fp                /* stream to test */
)
```

**DESCRIPTION** This routine tests the error indicator for the stream pointed to by *fp*.

**INCLUDE FILES** `stdio.h`

**RETURNS** Non-zero if the error indicator is set for *fp*.

**SEE ALSO** `ansiStdio`, `clearerr()`

---

## **fflush()**

**NAME** `fflush()` – flush a stream (ANSI)

**SYNOPSIS**

```
int fflush
(
    FILE * fp                /* stream to flush */
)
```

**DESCRIPTION** This routine writes to the file any unwritten data for a specified output or update stream for which the most recent operation was not input; for an input stream the behavior is undefined.

---

**WARNING:** ANSI specifies that if *fp* is a null pointer, `fflush()` performs the flushing action on all streams for which the behavior is defined; however, this is not implemented in VxWorks.

---

**INCLUDE FILES** `stdio.h`

**RETURNS** Zero, or EOF if a write error occurs.

**ERRNO** `EBADF`

**SEE ALSO** `ansiStdio`, `fclose()`

## fgetc()

<b>NAME</b>	<b>fgetc()</b> – return the next character from a stream (ANSI)
<b>SYNOPSIS</b>	<pre>int fgetc (     FILE * fp                /* stream to read from */ )</pre>
<b>DESCRIPTION</b>	<p>This routine returns the next character (converted to an <b>int</b>) from the specified stream, and advances the file position indicator for the stream.</p> <p>If the stream is at end-of-file, the end-of-file indicator for the stream is set; if a read error occurs, the error indicator is set.</p>
<b>INCLUDE FILES</b>	<b>stdio.h</b>
<b>RETURNS</b>	The next character from the stream, or EOF if the stream is at end-of-file or a read error occurs.
<b>SEE ALSO</b>	<b>ansiStdio, fgets(), getc()</b>

---

## fgetpos()

<b>NAME</b>	<b>fgetpos()</b> – store the current value of the file position indicator for a stream (ANSI)
<b>SYNOPSIS</b>	<pre>int fgetpos (     FILE * fp,                /* stream */     fpos_t * pos              /* where to store position */ )</pre>
<b>DESCRIPTION</b>	<p>This routine stores the current value of the file position indicator for a specified stream <i>fp</i> in the object pointed to by <i>pos</i>. The value stored contains unspecified information usable by <b>fsetpos()</b> for repositioning the stream to its position at the time <b>fgetpos()</b> was called.</p>
<b>INCLUDE FILES</b>	<b>stdio.h</b>
<b>RETURNS</b>	Zero, or non-zero if unsuccessful, with <b>errno</b> set to indicate the error.
<b>SEE ALSO</b>	<b>ansiStdio, fsetpos()</b>

---

## fgets()

**NAME** `fgets()` – read a specified number of characters from a stream (ANSI)

**SYNOPSIS**

```
char * fgets
(
    char * buf,                /* where to store characters */
    size_t n,                  /* no. of bytes to read + 1 */
    FILE * fp                   /* stream to read from */
)
```

**DESCRIPTION** This routine stores in the array *buf* up to *n*-1 characters from a specified stream. No additional characters are read after a new-line or end-of-line. A null character is written immediately after the last character read into the array.

If end-of-file is encountered and no characters have been read, the contents of the array remain unchanged. If a read error occurs, the array contents are indeterminate.

**INCLUDE FILES** `stdio.h`

**RETURNS** A pointer to *buf*, or a null pointer if an error occurs or end-of-file is encountered and no characters have been read.

**SEE ALSO** `ansiStdio`, `fread()`, `fgetc()`

---

## fileno()

**NAME** `fileno()` – return the file descriptor for a stream (POSIX)

**SYNOPSIS**

```
int fileno
(
    FILE * fp                   /* stream */
)
```

**DESCRIPTION** This routine returns the file descriptor associated with a specified stream.

**INCLUDE FILES** `stdio.h`

**RETURNS** The file descriptor, or -1 if an error occurs, with `errno` set to indicate the error.

**SEE ALSO** `ansiStdio`, *Information Technology - POSIX - Part 1: System API [C Language]*, *IEEE Std 1003.1*

---

## fileUploadPathClose()

<b>NAME</b>	<b>fileUploadPathClose()</b> – close the event-destination file (WindView)
<b>SYNOPSIS</b>	<pre>void fileUploadPathClose (     UPLOAD_ID pathId          /* generic upload-path descriptor */ )</pre>
<b>DESCRIPTION</b>	This routine closes the file associated with <i>pathId</i> that is serving as a destination for event data.
<b>RETURNS</b>	N/A
<b>SEE ALSO</b>	<b>wvFileUploadPathLib</b> , <b>fileUploadPathCreate()</b>

---

## fileUploadPathCreate()

<b>NAME</b>	<b>fileUploadPathCreate()</b> – create a file for depositing event data (Windview)
<b>SYNOPSIS</b>	<pre>UPLOAD_ID fileUploadPathCreate (     char * fname,            /* name of file to create */     int  openFlags          /* O_CREAT, O_TRUNC */ )</pre>
<b>DESCRIPTION</b>	This routine opens and initializes a file to receive uploaded events. The <i>openFlags</i> argument is passed on as the flags argument to the actual open call so that the caller can specify things like <b>O_TRUNC</b> and <b>O_CREAT</b> . The file is always opened as <b>O_WRONLY</b> , regardless of the value of <i>openFlags</i> .
<b>RETURNS</b>	The <b>UPLOAD_ID</b> , or <b>NULL</b> if the file can not be opened or memory for the ID is not available.
<b>SEE ALSO</b>	<b>wvFileUploadPathLib</b> , <b>fileUploadPathClose()</b>

---

## fileUploadPathLibInit( )

**NAME** `fileUploadPathLibInit()` – initialize the `wvFileUploadPathLib` library (Windview)

**SYNOPSIS** `STATUS fileUploadPathLibInit (void)`

**DESCRIPTION** This routine initializes the library by pulling in the routines in this file for use with WindView. It is called during system configuration from `usrWindview.c`.

**RETURNS** OK.

**SEE ALSO** `wvFileUploadPathLib`

---

## fileUploadPathWrite( )

**NAME** `fileUploadPathWrite()` – write to the event-destination file (WindView)

**SYNOPSIS**

```
int fileUploadPathWrite
(
    UPLOAD_ID pathId,          /* generic upload-path descriptor */
    char *    pStart,         /* address of data to write */
    size_t    size           /* number of bytes of data at pStart */
)
```

**DESCRIPTION** This routine writes *size* bytes of data beginning at *pStart* to the file indicated by *pathId*.

**RETURNS** The number of bytes written, or **ERROR**.

**SEE ALSO** `wvFileUploadPathLib`

---

## fioFormatV()

**NAME** `fioFormatV()` – convert a format string

**SYNOPSIS**

```
int fioFormatV
(
    const char * fmt,          /* format string */
    va_list    vaList,       /* pointer to varargs list */
    FUNCPTR    outRoutine,   /* handler for args as they're formatted */
    int        outarg        /* argument to routine */
)
```

**DESCRIPTION** This routine is used by the `printf()` family of routines to handle the actual conversion of a format string. The first argument is a format string, as described in the entry for `printf()`. The second argument is a variable argument list *vaList* that was previously established.

As the format string is processed, the result will be passed to the output routine whose address is passed as the third parameter, *outRoutine*. This output routine may output the result to a device, or put it in a buffer. In addition to the buffer and length to output, the fourth argument, *outarg*, will be passed through as the third parameter to the output routine. This parameter could be a file descriptor, a buffer address, or any other value that can be passed in an “int”.

The output routine should be declared as follows:

```
STATUS outRoutine
(
    char *buffer, /* buffer passed to routine */
    int  nchars, /* length of buffer */
    int  outarg  /* arbitrary arg passed to fmt routine */
)
```

The output routine should return **OK** if successful, or **ERROR** if unsuccessful.

**RETURNS** The number of characters output, or **ERROR** if the output routine returned **ERROR**.

**SEE ALSO** `fioLib`

---

## fiOLibInit()

<b>NAME</b>	<b>fiOLibInit()</b> – initialize the formatted I/O support library
<b>SYNOPSIS</b>	<pre>void fiOLibInit (void)</pre>
<b>DESCRIPTION</b>	This routine initializes the formatted I/O support library. It should be called once in <b>usrRoot()</b> when formatted I/O functions such as <b>printf()</b> and <b>scanf()</b> are used.
<b>RETURNS</b>	N/A
<b>SEE ALSO</b>	<b>fiOLib</b>

---

## fiORdString()

<b>NAME</b>	<b>fiORdString()</b> – read a string from a file
<b>SYNOPSIS</b>	<pre>int fiORdString (     int fd,                /* fd of device to read */     char string[],        /* buffer to receive input */     int maxbytes          /* max no. of chars to read */ )</pre>
<b>DESCRIPTION</b>	This routine puts a line of input into <i>string</i> . The specified input file descriptor is read until <i>maxbytes</i> , an EOF, an EOS, or a newline character is reached. A newline character or EOF is replaced with EOS, unless <i>maxbytes</i> characters have been read.
<b>RETURNS</b>	The length of the string read, including the terminating EOS; or EOF if a read error occurred or end-of-file occurred without reading any other character.
<b>SEE ALSO</b>	<b>fiOLib</b>

---

## fioread()

<b>NAME</b>	<b>fioread()</b> – read a buffer
<b>SYNOPSIS</b>	<pre>int fioread (     int    fd,                /* file descriptor of file to read */     char * buffer,           /* buffer to receive input */     int    maxbytes          /* maximum number of bytes to read */ )</pre>
<b>DESCRIPTION</b>	This routine repeatedly calls the routine <b>read()</b> until <i>maxbytes</i> have been read into <i>buffer</i> . If EOF is reached, the number of bytes read will be less than <i>maxbytes</i> .
<b>RETURNS</b>	The number of bytes read, or <b>ERROR</b> if there is an error during the read operation.
<b>SEE ALSO</b>	<b>fiolib</b> , <b>read()</b>

---

## floatInit()

<b>NAME</b>	<b>floatInit()</b> – initialize floating-point I/O support
<b>SYNOPSIS</b>	<pre>void floatInit (void)</pre>
<b>DESCRIPTION</b>	This routine must be called if floating-point format specifications are to be supported by the <b>printf()/scanf()</b> family of routines. If the configuration macro <b>INCLUDE_FLOATING_POINT</b> is defined, it is called by the root task, <b>usrRoot()</b> , in <b>usrConfig.c</b> .
<b>RETURNS</b>	N/A
<b>SEE ALSO</b>	<b>floatLib</b>

---

## floor( )

<b>NAME</b>	<b>floor( )</b> – compute the largest integer less than or equal to a specified value (ANSI)
<b>SYNOPSIS</b>	<pre>double floor (     double v          /* value to find the floor of */ )</pre>
<b>DESCRIPTION</b>	This routine returns the largest integer less than or equal to $v$ , in double precision.
<b>INCLUDE FILES</b>	<b>math.h</b>
<b>RETURNS</b>	The largest integral value less than or equal to $v$ , in double precision.
<b>SEE ALSO</b>	<b>ansiMath, mathALib</b>

---

## floorf( )

<b>NAME</b>	<b>floorf( )</b> – compute the largest integer less than or equal to a specified value (ANSI)
<b>SYNOPSIS</b>	<pre>float floorf (     float v          /* value to find the floor of */ )</pre>
<b>DESCRIPTION</b>	This routine returns the largest integer less than or equal to $v$ , in single precision.
<b>INCLUDE FILES</b>	<b>math.h</b>
<b>RETURNS</b>	The largest integral value less than or equal to $v$ , in single precision.
<b>SEE ALSO</b>	<b>mathALib</b>

## fmod()

<b>NAME</b>	<b>fmod()</b> – compute the remainder of $x/y$ (ANSI)
<b>SYNOPSIS</b>	<pre>double fmod (     double x,          /* numerator */     double y          /* denominator */ )</pre>
<b>DESCRIPTION</b>	This routine returns the remainder of $x/y$ with the sign of $x$ , in double precision.
<b>INCLUDE FILES</b>	<b>math.h</b>
<b>RETURNS</b>	The value $x - i * y$ , for some integer $i$ . If $y$ is non-zero, the result has the same sign as $x$ and magnitude less than the magnitude of $y$ . If $y$ is zero, <b>fmod()</b> returns zero.
<b>ERRNO</b>	EDOM
<b>SEE ALSO</b>	<b>ansiMath, mathALib</b>

---

## fmodf()

<b>NAME</b>	<b>fmodf()</b> – compute the remainder of $x/y$ (ANSI)
<b>SYNOPSIS</b>	<pre>float fmodf (     float x,  /* numerator */     float y  /* denominator */ )</pre>
<b>DESCRIPTION</b>	This routine returns the remainder of $x/y$ with the sign of $x$ , in single precision.
<b>INCLUDE FILES</b>	<b>math.h</b>
<b>RETURNS</b>	The single-precision modulus of $x/y$ .
<b>SEE ALSO</b>	<b>mathALib</b>

---

## fopen()

<b>NAME</b>	<b>fopen()</b> – open a file specified by name (ANSI)
<b>SYNOPSIS</b>	<pre><b>FILE * fopen</b>   (     <b>const char * file,</b>          /* name of file */     <b>const char * mode</b>         /* mode */   )</pre>
<b>DESCRIPTION</b>	<p>This routine opens a file whose name is the string pointed to by <i>file</i> and associates a stream with it. The argument <i>mode</i> points to a string beginning with one of the following sequences:</p> <ul style="list-style-type: none"><li>r open text file for reading</li><li>w truncate to zero length or create text file for writing</li><li>a append; open or create text file for writing at end-of-file</li><li>rb open binary file for reading</li><li>wb truncate to zero length or create binary file for writing</li><li>ab append; open or create binary file for writing at end-of-file</li><li>r+ open text file for update (reading and writing)</li><li>w+ truncate to zero length or create text file for update.</li><li>a+ append; open or create text file for update, writing at end-of-file</li><li>r+b / rb+ open binary file for update (reading and writing)</li><li>w+b / wb+ truncate to zero length or create binary file for update</li><li>a+b / ab+ append; open or create binary file for update, writing at end-of-file</li></ul>

Opening a file with read mode (**r** as the first character in the *mode* argument) fails if the file does not exist or cannot be read.

Opening a file with append mode (**a** as the first character in the *mode* argument) causes all subsequent writes to the file to be forced to the then current end-of-file, regardless of intervening calls to **fseek()**. In some implementations, opening a binary file with append mode (**b** as the second or third character in the *mode* argument) may initially position the file position indicator for the stream beyond the last data written, because of null character padding. In VxWorks, whether append mode is supported is device-specific.

When a file is opened with update mode (**+** as the second or third character in the *mode* argument), both input and output may be performed on the associated stream. However, output may not be directly followed by input without an intervening call to **fflush()** or to a file positioning function (**fseek()**, **fsetpos()**, or **rewind()**), and input may not be directly followed by output without an intervening call to a file positioning function, unless the input operation encounters end-of-file. Opening (or creating) a text file with update mode may instead open (or create) a binary stream in some implementations.

When opened, a stream is fully buffered if and only if it can be determined not to refer to an interactive device. The error and end-of-file indicators for the stream are cleared.

<b>INCLUDE FILES</b>	<b>stdio.h</b>
<b>RETURNS</b>	A pointer to the object controlling the stream, or a null pointer if the operation fails.
<b>SEE ALSO</b>	<b>ansiStdio</b> , <b>fdopen()</b> , <b>freopen()</b>

---

## fppInit()

<b>NAME</b>	<b>fppInit()</b> – initialize floating-point coprocessor support
<b>SYNOPSIS</b>	<b>void fppInit (void)</b>
<b>DESCRIPTION</b>	This routine initializes floating-point coprocessor support and must be called before using the floating-point coprocessor. This is done automatically by the root task, <b>usrRoot()</b> , in <b>usrConfig.c</b> when the configuration macro <b>INCLUDE_HW_FP</b> is defined.
<b>RETURNS</b>	N/A
<b>SEE ALSO</b>	<b>fppLib</b>

---

## fppProbe()

<b>NAME</b>	<b>fppProbe()</b> – probe for the presence of a floating-point coprocessor
<b>SYNOPSIS</b>	<b>STATUS</b> fppProbe (void)
<b>DESCRIPTION</b>	<p>This routine determines whether there is a floating-point coprocessor in the system. The implementation of this routine is architecture-dependent:</p> <p><b>MC680x0, x86, SH-4:</b> This routine sets the illegal coprocessor opcode trap vector and executes a coprocessor instruction. If the instruction causes an exception, <b>fppProbe()</b> returns <b>ERROR</b>. Note that this routine saves and restores the illegal coprocessor opcode trap vector that was there prior to this call.</p> <p>The probe is only performed the first time this routine is called. The result is stored in a static and returned on subsequent calls without actually probing.</p> <p><b>MIPS:</b> This routine simply reads the R-Series status register and reports the bit that indicates whether coprocessor 1 is usable. This bit must be correctly initialized in the BSP.</p> <p><b>ARM:</b> This routine currently returns <b>ERROR</b> to indicate no floating-point coprocessor support.</p> <p><b>SimNT, SimSolaris:</b> This routine currently returns <b>OK</b>.</p>
<b>RETURNS</b>	OK, or <b>ERROR</b> if there is no floating-point coprocessor.
<b>SEE ALSO</b>	fppArchLib

---

## fppRestore()

<b>NAME</b>	<b>fppRestore()</b> – restore the floating-point coprocessor context
<b>SYNOPSIS</b>	<pre>void fppRestore (     FP_CONTEXT * pFpContext    /* where to restore context from */ )</pre>

**DESCRIPTION** This routine restores the floating-point coprocessor context. The context restored is:

**MC680x0:**

- registers **fpcr**, **fpsr**, and **fpiar**
- registers **f0** - **f7**
- internal state frame (if **NULL**, the other registers are not saved.)

**MIPS:**

- register **fpcsr**
- registers **fp0** - **fp31**

**SH-4:**

- registers **fpcsr** and **fpul**
- registers **fr0** - **fr15**
- registers **xf0** - **xf15**

**x86:**

108 byte old context with **fsave** and **frstor** instruction

- control word, status word, tag word,
- instruction pointer,
- instruction pointer selector,
- last FP instruction op code,
- data pointer,
- data pointer selector,
- registers **st/mm0** - **st/mm7** (10 bytes \* 8)

512 byte new context with **fxsave** and **fxrstor** instruction

- control word, status word, tag word,
- last FP instruction op code,
- instruction pointer,
- instruction pointer selector,
- data pointer,
- data pointer selector,
- registers **st/mm0** - **st/mm7** (10 bytes \* 8)
- registers **xmm0** - **xmm7** (16 bytes \* 8)

**ARM:**

- currently, on this architecture, this routine does nothing.

**SimSolaris:**

- register **fsr**
- registers **f0** - **f31**

**SimNT:**

- this routine does nothing on Windows simulator.

**RETURNS** N/A

**SEE ALSO** **fppArchLib**, **fppSave()**

---

## fppSave()

- NAME** `fppSave()` – save the floating-point coprocessor context
- SYNOPSIS**
- ```
void fppSave
(
    FP_CONTEXT * pFpContext /* where to save context */
)
```
- DESCRIPTION** This routine saves the floating-point coprocessor context. The context saved is:
- MC680x0:**
- registers `fpcr`, `fpsr`, and `fpiar`
  - registers `f0` - `f7`
  - internal state frame (if `NULL`, the other registers are not saved.)
- MIPS:**
- register `fpcsr`
  - registers `fp0` - `fp31`
- SH-4:**
- registers `fpcsr` and `fpul`
  - registers `fr0` - `fr15`
  - registers `xf0` - `xf15`
- x86:**
- 108 byte old context with `fsave` and `frstor` instruction
- control word, status word, tag word,
  - instruction pointer,
  - instruction pointer selector,
  - last FP instruction op code,
  - data pointer,
  - data pointer selector,
  - registers `st/mm0` - `st/mm7` (10 bytes \* 8)
- 512 byte new context with `fxsave` and `fxrstor` instruction
- control word, status word, tag word,
  - last FP instruction op code,
  - instruction pointer,
  - instruction pointer selector,
  - data pointer,
  - data pointer selector,
  - registers `st/mm0` - `st/mm7` (10 bytes \* 8)
  - registers `xmm0` - `xmm7` (16 bytes \* 8)

**ARM:**

- currently, on this architecture, this routine does nothing.

**SimSolaris:**

- register **fsr**
- registers **f0 - f31**

**SimNT:**

- this routine does nothing on Windows simulator. Floating point registers are saved by Windows.

**RETURNS** N/A

**SEE ALSO** fppArchLib, fppRestore()

---

## fppShowInit()

**NAME** fppShowInit() – initialize the floating-point show facility

**SYNOPSIS** void fppShowInit (void)

**DESCRIPTION** This routine links the floating-point show facility into the VxWorks system. It is called automatically when the floating-point show facility is configured into VxWorks using either of the following methods:

– If you use the configuration header files, define

**INCLUDE\_SHOW\_ROUTINES** in **config.h**.

– If you use the Tornado project facility, select **INCLUDE\_HW\_FP\_SHOW**.

**RETURNS** N/A

**SEE ALSO** fppShow

---

## fppTaskRegsGet()

**NAME** fppTaskRegsGet() – get the floating-point registers from a task TCB

**SYNOPSIS**

```
STATUS fppTaskRegsGet
(
    int          task,          /* task to get info about */
    FPREG_SET * pFpRegSet     /* ptr to floating-point register set */
)
```

**DESCRIPTION** This routine copies a task's floating-point registers and/or status registers to the locations whose pointers are passed as parameters. The floating-point registers are copied into an array containing all the registers.

---

**NOTE:** This routine only works well if *task* is not the calling task. If a task tries to discover its own registers, the values will be stale (that is, left over from the last task switch).

---

**RETURNS** OK, or ERROR if there is no floating-point support or there is an invalid state.

**SEE ALSO** fppArchLib, fppTaskRegsSet()

---

## fppTaskRegsSet()

**NAME** fppTaskRegsSet() – set the floating-point registers of a task

**SYNOPSIS**

```
STATUS fppTaskRegsSet
(
    int          task,          /* task to set registers for */
    FPREG_SET * pFpRegSet     /* ptr to floating-point register set */
)
```

**DESCRIPTION** This routine loads the specified values into the TCB of a specified task. The register values are copied from the array at *pFpRegSet*.

**RETURNS** OK, or ERROR if there is no floating-point support or there is an invalid state.

**SEE ALSO** fppArchLib, fppTaskRegsGet()

---

## fppTaskRegsShow()

|                    |                                                                                                              |
|--------------------|--------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>        | <b>fppTaskRegsShow()</b> – print the contents of a task’s floating-point registers                           |
| <b>SYNOPSIS</b>    | <pre>void fppTaskRegsShow (     int task          /* task to display floating point registers for */ )</pre> |
| <b>DESCRIPTION</b> | This routine prints to standard output the contents of a task’s floating-point registers.                    |
| <b>RETURNS</b>     | N/A                                                                                                          |
| <b>SEE ALSO</b>    | <b>fppShow</b>                                                                                               |

---

## fprintf()

|                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
|--------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>        | <b>fprintf()</b> – write a formatted string to a stream (ANSI)                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| <b>SYNOPSIS</b>    | <pre>int fprintf (     FILE *    fp,          /* stream to write to */     const char * fmt,     /* format string */     ...      /* optional arguments to format string */ )</pre>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| <b>DESCRIPTION</b> | <p>This routine writes output to a specified stream under control of the string <i>fmt</i>. The string <i>fmt</i> contains ordinary characters, which are written unchanged, plus conversion specifications, which cause the arguments that follow <i>fmt</i> to be converted and printed as part of the formatted string.</p> <p>The number of arguments for the format is arbitrary, but they must correspond to the conversion specifications in <i>fmt</i>. If there are insufficient arguments, the behavior is undefined. If the format is exhausted while arguments remain, the excess arguments are evaluated but otherwise ignored. The routine returns when the end of the format string is encountered.</p> <p>The format is a multibyte character sequence, beginning and ending in its initial shift state. The format is composed of zero or more directives: ordinary multibyte characters (not %) that are copied unchanged to the output stream; and conversion specification, each of which results in fetching zero or more subsequent arguments. Each conversion</p> |

specification is introduced by the % character. After the %, the following appear in sequence:

- Zero or more flags (in any order) that modify the meaning of the conversion specification.
- An optional minimum field width. If the converted value has fewer characters than the field width, it will be padded with spaces (by default) on the left (or right, if the left adjustment flag, described later, has been given) to the field width. The field width takes the form of an asterisk (\*) (described later) or a decimal integer.
- An optional precision that gives the minimum number of digits to appear for the **d**, **i**, **o**, **u**, **x**, and **X** conversions, the number of digits to appear after the decimal-point character for **e**, **E**, and **f** conversions, the maximum number of significant digits for the **g** and **G** conversions, or the maximum number of characters to be written from a string in the **s** conversion. The precision takes the form of a period (.) followed either by an asterisk (\*) (described later) or by an optional decimal integer; if only the period is specified, the precision is taken as zero. If a precision appears with any other conversion specifier, the behavior is undefined.
- An optional **h** specifying that a following **d**, **i**, **o**, **u**, **x**, and **X** conversion specifier applies to a **short int** or **unsigned short int** argument (the argument will have been promoted according to the integral promotions, and its value converted to **short int** or **unsigned short int** before printing); an optional **h** specifying that a following **n** conversion specifier applies to a pointer to a **short int** argument; an optional **l** (el) specifying that a following **d**, **i**, **o**, **u**, **x**, and **X** conversion specifier applies to a **long int** or **unsigned long int** argument; or an optional **l** specifying that a following **n** conversion specifier applies to a pointer to a **long int** argument. If an **h** or **l** appears with any other conversion specifier, the behavior is undefined.

---

**WARNING:** ANSI C also specifies an optional **L** in some of the same contexts as **l** above, corresponding to a **long double** argument. However, the current release of the VxWorks libraries does not support **long double** data; using the optional **L** gives unpredictable results.

---

- A character that specifies the type of conversion to be applied.

As noted above, a field width, or precision, or both, can be indicated by an asterisk (\*). In this case, an **int** argument supplies the field width or precision. The arguments specifying field width, or precision, or both, should appear (in that order) before the argument (if any) to be converted. A negative field width argument is taken as a - flag followed by a positive field width. A negative precision argument is taken as if the precision were omitted.

The flag characters and their meanings are:

-

The result of the conversion will be left-justified within the field. (it will be right-justified if this flag is not specified.)

**fprintf()**

+

The result of a signed conversion will always begin with a plus or minus sign. (It will begin with a sign only when a negative value is converted if this flag is not specified.)

space

If the first character of a signed conversion is not a sign, or if a signed conversion results in no characters, a space will be prefixed to the result. If the **space** and **+** flags both appear, the **space** flag will be ignored.

#

The result is to be converted to an “alternate form.” For **o** conversion it increases the precision to force the first digit of the result to be a zero. For **x** (or **X**) conversion, a non-zero result will have “0x” (or “0X”) prefixed to it. For **e**, **E**, **f**, **g**, and **G** conversions, the result will always contain a decimal-point character, even if no digits follow it. (Normally, a decimal-point character appears in the result of these conversions only if no digit follows it). For **g** and **G** conversions, trailing zeros will not be removed from the result. For other conversions, the behavior is undefined.

0

For **d**, **i**, **o**, **u**, **x**, **X**, **e**, **E**, **f**, **g**, and **G** conversions, leading zeros (following any indication of sign or base) are used to pad to the field width; no space padding is performed. If the **0** and **-** flags both appear, the **0** flag will be ignored. For **d**, **i**, **o**, **u**, **x**, and **X** conversions, if a precision is specified, the **0** flag will be ignored. For other conversions, the behavior is undefined.

The conversion specifiers and their meanings are:

**d, i**

The **int** argument is converted to signed decimal in the style **[-]dddd**. The precision specifies the minimum number of digits to appear; if the value being converted can be represented in fewer digits, it will be expanded with leading zeros. The default precision is 1. The result of converting a zero value with a precision of zero is no characters.

**o, u, x, X**

The **unsigned int** argument is converted to unsigned octal (**o**), unsigned decimal (**u**), or unsigned hexadecimal notation (**x** or **X**) in the style **dddd**; the letters abcdef are used for **x** conversion and the letters ABCDEF for **X** conversion. The precision specifies the minimum number of digits to appear; if the value being converted can be represented in fewer digits, it will be expanded with leading zeros. The default precision is 1. The result of converting a zero value with a precision of zero is no characters.

**f**

The **double** argument is converted to decimal notation in the style **[-]ddd.ddd**, where the number of digits after the decimal point character is equal to the precision specification. If the precision is missing, it is taken as 6; if the precision is zero and the **#** flag is not specified, no decimal-point character appears. If a decimal-point character appears, at least one digit appears before it. The value is rounded to the

appropriate number of digits.

**e, E**

The **double** argument is converted in the style `[-]d.ddde+/-dd`, where there is one digit before the decimal-point character (which is non-zero if the argument is non-zero) and the number of digits after it is equal to the precision; if the precision is missing, it is taken as 6; if the precision is zero and the **#** flag is not specified, no decimal-point character appears. The value is rounded to the appropriate number of digits. The **E** conversion specifier will produce a number with **E** instead of **e** introducing the exponent. The exponent always contains at least two digits. If the value is zero, the exponent is zero.

**g, G**

The **double** argument is converted in style **f** or **e** (or in style **E** in the case of a **G** conversion specifier), with the precision specifying the number of significant digits. If the precision is zero, it is taken as 1. The style used depends on the value converted; style **e** (or **E**) will be used only if the exponent resulting from such a conversion is less than -4 or greater than or equal to the precision. Trailing zeros are removed from the fractional portion of the result; a decimal-point character appears only if it is followed by a digit.

**c**

The **int** argument is converted to an **unsigned char**, and the resulting character is written.

**s**

The argument should be a pointer to an array of character type. Characters from the array are written up to (but not including) a terminating null character; if the precision is specified, no more than that many characters are written. If the precision is not specified or is greater than the size of the array, the array will contain a null character.

**p**

The argument should be a pointer to **void**. The value of the pointer is converted to a sequence of printable characters, in hexadecimal representation (prefixed with "0x").

**n**

The argument should be a pointer to an integer into which the number of characters written to the output stream so far by this call to **fprintf()** is written. No argument is converted.

**%**

A **%** is written. No argument is converted. The complete conversion specification is **%%**.

If a conversion specification is invalid, the behavior is undefined.

If any argument is, or points to, a union or an aggregate (except for an array of character type using **s** conversion, or a pointer using **p** conversion), the behavior is undefined.

**fputc( )**

In no case does a non-existent or small field width cause truncation of a field if the result of a conversion is wider than the field width, the field is expanded to contain the conversion result.

|                      |                                                                                  |
|----------------------|----------------------------------------------------------------------------------|
| <b>INCLUDE FILES</b> | <b>stdio.h</b>                                                                   |
| <b>RETURNS</b>       | The number of characters written, or a negative value if an output error occurs. |
| <b>SEE ALSO</b>      | <b>ansiStdio, printf( )</b>                                                      |

---

## **fputc( )**

|                      |                                                                                                                                                                                                                                                                                                                                                      |
|----------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>          | <b>fputc( )</b> – write a character to a stream (ANSI)                                                                                                                                                                                                                                                                                               |
| <b>SYNOPSIS</b>      | <pre> int fputc (     int    c,                /* character to write */     FILE * fp                /* stream to write to */ ) </pre>                                                                                                                                                                                                               |
| <b>DESCRIPTION</b>   | <p>This routine writes a character <i>c</i> to a specified stream, at the position indicated by the stream's file position indicator (if defined), and advances the indicator appropriately.</p> <p>If the file cannot support positioning requests, or if the stream was opened in append mode, the character is appended to the output stream.</p> |
| <b>INCLUDE FILES</b> | <b>stdio.h</b>                                                                                                                                                                                                                                                                                                                                       |
| <b>RETURNS</b>       | The character written, or EOF if a write error occurs, with the error indicator set for the stream.                                                                                                                                                                                                                                                  |
| <b>SEE ALSO</b>      | <b>ansiStdio, fputs( ), putc( )</b>                                                                                                                                                                                                                                                                                                                  |

---

## fputs()

**NAME** fputs() – write a string to a stream (ANSI)

**SYNOPSIS**

```
int fputs
(
    const char * s,          /* string */
    FILE *      fp          /* stream to write to */
)
```

**DESCRIPTION** This routine writes the string *s*, minus the terminating NULL character, to a specified stream.

**INCLUDE FILES** stdio.h

**RETURNS** A non-negative value, or EOF if a write error occurs.

**SEE ALSO** ansiStdio, fputc()

---

## fread()

**NAME** fread() – read data into an array (ANSI)

**SYNOPSIS**

```
int fread
(
    void * buf,              /* where to copy data */
    size_t size,            /* element size */
    size_t count,          /* no. of elements */
    FILE * fp               /* stream to read from */
)
```

**DESCRIPTION** This routine reads, into the array *buf*, up to *count* elements of size *size*, from a specified stream *fp*. The file position indicator for the stream (if defined) is advanced by the number of characters successfully read. If an error occurs, the resulting value of the file position indicator for the stream is indeterminate. If a partial element is read, its value is indeterminate.

**INCLUDE FILES** stdio.h

**RETURNS** The number of elements successfully read, which may be less than *count* if a read error or end-of-file is encountered; or zero if *size* or *count* is zero, with the contents of the array and the state of the stream remaining unchanged.

**SEE ALSO** **ansiStdio**

---

## free()

**NAME** **free()** – free a block of memory (ANSI)

**SYNOPSIS**

```
void free
(
    void * ptr          /* pointer to block of memory to free */
)
```

**DESCRIPTION** This routine returns to the free memory pool a block of memory previously allocated with **malloc()** or **calloc()**.

**RETURNS** N/A

**SEE ALSO** **memPartLib**, **malloc()**, **calloc()**, *American National Standard for Information Systems - Programming Language - C, ANSI X3.159-1989: General Utilities (stdlib.h)*

---

## freopen()

**NAME** **freopen()** – open a file specified by name (ANSI)

**SYNOPSIS**

```
FILE * freopen
(
    const char * file,    /* name of file */
    const char * mode,    /* mode */
    FILE *      fp        /* stream */
)
```

**DESCRIPTION** This routine opens a file whose name is the string pointed to by *file* and associates it with a specified stream *fp*. The *mode* argument is used just as in the **fopen()** function.

This routine first attempts to close any file that is associated with the specified stream. Failure to close the file successfully is ignored. The error and end-of-file indicators for the stream are cleared.

Typically, **freopen()** is used to attach the already-open streams **stdin**, **stdout**, and **stderr** to other files.

**INCLUDE FILES**     **stdio.h**

**RETURNS**            The value of *fp*, or a null pointer if the open operation fails.

**SEE ALSO**            **ansiStdio, fopen()**

---

## **frexp()**

**NAME**                **frexp()** – break a floating-point number into a normalized fraction and power of 2 (ANSI)

**SYNOPSIS**            **double frexp**  
                      (  
                      **double value,**                /\* number to be normalized \*/  
                      **int \* pexp**                    /\* pointer to the exponent \*/  
                      )

**DESCRIPTION**        This routine breaks a double-precision number *value* into a normalized fraction and integral power of 2. It stores the integer exponent in *pexp*.

**INCLUDE FILES**     **math.h**

**RETURNS**            The double-precision value *x*, such that the magnitude of *x* is in the interval  $[1/2,1)$  or zero, and *value* equals *x* times 2 to the power of *pexp*. If *value* is zero, both parts of the result are zero.

**ERRNO**              **EDOM**

**SEE ALSO**            **ansiMath**

---

## fscanf()

**NAME** fscanf() – read and convert characters from a stream (ANSI)

**SYNOPSIS**

```
int fscanf
(
    FILE *      fp,          /* stream to read from */
    char const * fmt,       /* format string */
    ...         /* arguments to format string */
)
```

**DESCRIPTION** This routine reads characters from a specified stream, and interprets them according to format specifications in the string *fmt*, which specifies the admissible input sequences and how they are to be converted for assignment, using subsequent arguments as pointers to the objects to receive the converted input.

If there are insufficient arguments for the format, the behavior is undefined. If the format is exhausted while arguments remain, the excess arguments are evaluated but are otherwise ignored.

The format is a multibyte character sequence, beginning and ending in its initial shift state. The format is composed of zero or more directives: one or more white-space characters; an ordinary multibyte character (neither % nor a white-space character); or a conversion specification. Each conversion specification is introduced by the % character. After the %, the following appear in sequence:

- An optional assignment-suppressing character \*.
- An optional non-zero decimal integer that specifies the maximum field width.
- An optional **h** or **l** (el) indicating the size of the receiving object. The conversion specifiers **d**, **i**, and **n** should be preceded by **h** if the corresponding argument is a pointer to **short int** rather than a pointer to **int**, or by **l** if it is a pointer to **long int**. Similarly, the conversion specifiers **o**, **u**, and **x** shall be preceded by **h** if the corresponding argument is a pointer to **unsigned short int** rather than a pointer to **unsigned int**, or by **l** if it is a pointer to **unsigned long int**. Finally, the conversion specifiers **e**, **f**, and **g** shall be preceded by **l** if the corresponding argument is a pointer to **double** rather than a pointer to **float**. If an **h** or **l** appears with any other conversion specifier, the behavior is undefined.

---

**WARNING:** ANSI C also specifies an optional **L** in some of the same contexts as **l** above, corresponding to a **long double \*** argument. However, the current release of the VxWorks libraries does not support **long double** data; using the optional **L** gives unpredictable results.

---

- A character that specifies the type of conversion to be applied. The valid conversion

specifiers are described below.

The **fscanf()** routine executes each directive of the format in turn. If a directive fails, as detailed below, **fscanf()** returns. Failures are described as input failures (due to the unavailability of input characters), or matching failures (due to inappropriate input).

A directive composed of white-space character(s) is executed by reading input up to the first non-white-space character (which remains unread), or until no more characters can be read.

A directive that is an ordinary multibyte character is executed by reading the next characters of the stream. If one of the characters differs from one comprising the directive, the directive fails, and the differing and subsequent characters remain unread.

A directive that is a conversion specification defines a set of matching input sequences, as described below for each specifier. A conversion specification is executed in the following steps:

Input white-space characters (as specified by the **isspace()** function) are skipped, unless the specification includes a **[**, **c**, or **n** specifier.

An input item is read from the stream, unless the specification includes an **n** specifier. An input item is defined as the longest matching sequence of input characters, unless that exceeds a specified field width, in which case it is the initial subsequence of that length in the sequence. The first character, if any, after the input item remains unread. If the length of the input item is zero, the execution of the directive fails: this condition is a matching failure, unless an error prevented input from the stream, in which case it is an input failure.

Except in the case of a **%** specifier, the input item is converted to a type appropriate to the conversion specifier. If the input item is not a matching sequence, the execution of the directive fails: this condition is a matching failure. Unless assignment suppression was indicated by a **\***, the result of the conversion is placed in the object pointed to by the first argument following the *fmt* argument that has not already received a conversion result. If this object does not have an appropriate type, or if the result of the conversion cannot be represented in the space provided, the behavior is undefined.

The following conversion specifiers are valid:

**d**

Matches an optionally signed decimal integer whose format is the same as expected for the subject sequence of the **strtol()** function with the value 10 for the *base* argument. The corresponding argument should be a pointer to **int**.

**i**

Matches an optionally signed integer, whose format is the same as expected for the subject sequence of the **strtol()** function with the value 0 for the *base* argument. The corresponding argument should be a pointer to **int**.

**o**

Matches an optionally signed octal integer, whose format is the same as expected for

**fscanf()**

the subject sequence of the **strtoul()** function with the value 8 for the *base* argument. The corresponding argument should be a pointer to **unsigned int**.

**u**

Matches an optionally signed decimal integer, whose format is the same as expected for the subject sequence of the **strtoul()** function with the value 10 for the *base* argument. The corresponding argument should be a pointer to **unsigned int**.

**x**

Matches an optionally signed hexadecimal integer, whose format is the same as expected for the subject sequence of the **strtoul()** function with the value 16 for the *base* argument. The corresponding argument should be a pointer to **unsigned int**.

**e, f, g**

Match an optionally signed floating-point number, whose format is the same as expected for the subject string of the **strtod()** function. The corresponding argument should be a pointer to **float**.

**s**

Matches a sequence of non-white-space characters. The corresponding argument should be a pointer to the initial character of an array large enough to accept the sequence and a terminating null character, which will be added automatically.

**[**

Matches a non-empty sequence of characters from a set of expected characters (the **scanset**). The corresponding argument should be a pointer to the initial character of an array large enough to accept the sequence and a terminating null character, which is added automatically. The conversion specifier includes all subsequent character in the format string, up to and including the matching right bracket (**]**). The characters between the brackets (the **scanlist**) comprise the scanset, unless the character after the left bracket is a circumflex (**^**) in which case the scanset contains all characters that do not appear in the scanlist between the circumflex and the right bracket. If the conversion specifier begins with **"["** or **"[^"**, the right bracket character is in the scanlist and the next right bracket character is the matching right bracket that ends the specification; otherwise the first right bracket character is the one that ends the specification.

**c**

Matches a sequence of characters of the number specified by the field width (1 if no field width is present in the directive). The corresponding argument should be a pointer to the initial character of an array large enough to accept the sequence. No null character is added.

**p**

Matches an implementation-defined set of sequences, which should be the same as the set of sequences that may be produced by the **%p** conversion of the **fprintf()** function. The corresponding argument should be a pointer to a pointer to **void**. VxWorks defines its pointer input field to be consistent with pointers written by the **fprintf()** function ("**0x**" hexadecimal notation). If the input item is a value converted

earlier during the same program execution, the pointer that results should compare equal to that value; otherwise the behavior of the %p conversion is undefined.

**n**

No input is consumed. The corresponding argument should be a pointer to **int** into which the number of characters read from the input stream so far by this call to **fscanf()** is written. Execution of a %n directive does not increment the assignment count returned when **fscanf()** completes execution.

**%**

Matches a single %; no conversion or assignment occurs. The complete conversion specification is %%. F

If a conversion specification is invalid, the behavior is undefined.

The conversion specifiers **E**, **G**, and **X** are also valid and behave the same as **e**, **g**, and **x**, respectively.

If end-of-file is encountered during input, conversion is terminated. If end-of-file occurs before any characters matching the current directive have been read (other than leading white space, where permitted), execution of the current directive terminates with an input failure; otherwise, unless execution of the current directive is terminated with a matching failure, execution of the following directive (if any) is terminated with an input failure.

If conversion terminates on a conflicting input character, the offending input character is left unread in the input stream. Trailing white space (including new-line characters) is left unread unless matched by a directive. The success of literal matches and suppressed assignments is not directly determinable other than via the %n directive.

**INCLUDE FILES**

**stdio.h**

**RETURNS**

The number of input items assigned, which can be fewer than provided for, or even zero, in the event of an early matching failure; or EOF if an input failure occurs before any conversion.

**SEE ALSO**

**ansiStdio**, **scanf()**, **sscanf()**

---

## fseek()

**NAME** `fseek()` – set the file position indicator for a stream (ANSI)

**SYNOPSIS**

```
int fseek
(
    FILE * fp,           /* stream */
    long  offset,       /* offset from whence */
    int   whence        /* position to offset from: SEEK_SET = */
                        /* beginning SEEK_CUR = current position */
                        /* SEEK_END = end-of-file */
)
```

**DESCRIPTION** This routine sets the file position indicator for a specified stream. For a binary stream, the new position, measured in characters from the beginning of the file, is obtained by adding *offset* to the position specified by *whence*, whose possible values are:

**SEEK\_SET**  
the beginning of the file.

**SEEK\_CUR**  
the current value of the file position indicator.

**SEEK\_END**  
the end of the file.

A binary stream does not meaningfully support `fseek()` calls with a *whence* value of **SEEK\_END**.

For a text stream, either *offset* is zero, or *offset* is a value returned by an earlier call to `ftell()` on the stream, in which case *whence* should be **SEEK\_SET**.

A successful call to `fseek()` clears the end-of-file indicator for the stream and undoes any effects of `ungetc()` on the same stream. After an `fseek()` call, the next operation on an update stream can be either input or output.

**INCLUDE FILES** `stdio.h`

**RETURNS** Non-zero only for a request that cannot be satisfied.

**ERRNO** `EINVAL`

**SEE ALSO** `ansiStdio`, `ftell()`

---

## fsetpos()

|                      |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
|----------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>          | <b>fsetpos()</b> – set the file position indicator for a stream (ANSI)                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| <b>SYNOPSIS</b>      | <pre>int fsetpos (     FILE *      iop,          /* stream */     const fpos_t * pos       /* position, obtained by fgetpos() */ )</pre>                                                                                                                                                                                                                                                                                                                                                                      |
| <b>DESCRIPTION</b>   | <p>This routine sets the file position indicator for a specified stream <i>iop</i> according to the value of the object pointed to by <i>pos</i>, which is a value obtained from an earlier call to <b>fgetpos()</b> on the same stream.</p> <p>A successful call to <b>fsetpos()</b> clears the end-of-file indicator for the stream and undoes any effects of <b>ungetc()</b> on the same stream. After an <b>fsetpos()</b> call, the next operation on an update stream may be either input or output.</p> |
| <b>INCLUDE FILES</b> | <b>stdio.h</b>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
| <b>RETURNS</b>       | Zero, or non-zero if the call fails, with <b>errno</b> set to indicate the error.                                                                                                                                                                                                                                                                                                                                                                                                                             |
| <b>SEE ALSO</b>      | <b>ansiStdio</b> , <b>fgetpos()</b>                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |

---

## fstat()

|                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
|--------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>        | <b>fstat()</b> – get file status information (POSIX)                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| <b>SYNOPSIS</b>    | <pre>STATUS fstat (     int          fd,          /* file descriptor for file to check */     struct stat * pStat      /* pointer to stat structure */ )</pre>                                                                                                                                                                                                                                                                                                                                                                                |
| <b>DESCRIPTION</b> | <p>This routine obtains various characteristics of a file (or directory). The file must already have been opened using <b>open()</b> or <b>creat()</b>. The <i>fd</i> parameter is the file descriptor returned by <b>open()</b> or <b>creat()</b>.</p> <p>The <i>pStat</i> parameter is a pointer to a <b>stat</b> structure (defined in <b>stat.h</b>). This structure must be allocated before <b>fstat()</b> is called.</p> <p>On return, fields in the <b>stat</b> structure are updated to reflect the characteristics of the file.</p> |

**fstatfs()**

**RETURNS** OK or ERROR.

**SEE ALSO** `dirLib`, `stat()`, `ls()`

---

**fstatfs()**

**NAME** `fstatfs()` – get file status information (POSIX)

**SYNOPSIS**

```
STATUS fstatfs
(
    int          fd,          /* file descriptor for file to check */
    struct statfs * pStat     /* pointer to statfs structure */
)
```

**DESCRIPTION** This routine obtains various characteristics of a file system. A file in the file system must already have been opened using `open()` or `creat()`. The *fd* parameter is the file descriptor returned by `open()` or `creat()`.

The *pStat* parameter is a pointer to a **statfs** structure (defined in `stat.h`). This structure must be allocated before `fstat()` is called.

Upon return, the fields in the **statfs** structure are updated to reflect the characteristics of the file.

**RETURNS** OK or ERROR.

**SEE ALSO** `dirLib`, `statfs()`, `ls()`

---

**ftell()**

**NAME** `ftell()` – return the current value of the file position indicator for a stream (ANSI)

**SYNOPSIS**

```
long ftell
(
    FILE * fp                /* stream */
)
```

**DESCRIPTION** This routine returns the current value of the file position indicator for a specified stream. For a binary stream, the value is the number of characters from the beginning of the file. For a text stream, the file position indicator contains unspecified information, usable by

**fseek()** for returning the file position indicator to its position at the time of the **ftell()** call; the difference between two such return values is not necessarily a meaningful measure of the number of characters written or read.

|                      |                                                                                                                        |
|----------------------|------------------------------------------------------------------------------------------------------------------------|
| <b>INCLUDE FILES</b> | <b>stdio.h</b>                                                                                                         |
| <b>RETURNS</b>       | The current value of the file position indicator, or -1L if unsuccessful, with <b>errno</b> set to indicate the error. |
| <b>SEE ALSO</b>      | <b>ansiStdio, fseek()</b>                                                                                              |

**F**

---

## ftpCommand()

**NAME** **ftpCommand()** – send an FTP command and get the reply

**SYNOPSIS**

```
int ftpCommand
(
    int    ctrlSock,          /* fd of control connection socket */
    char * fmt,              /* format string of command to send */
    int    arg1,             /* first of six args to format string */
    int    arg2,
    int    arg3,
    int    arg4,
    int    arg5,
    int    arg6
)
```

**DESCRIPTION** This command has been superseded by **ftpCommandEnhanced()**

This routine sends the specified command on the specified socket, which should be a control connection to a remote FTP server. The command is specified as a string in **printf()** format with up to six arguments.

After the command is sent, **ftpCommand()** waits for the reply from the remote server. The FTP reply code is returned in the same way as in **ftpReplyGet()**.

**EXAMPLE**

```
ftpCommand (ctrlSock, "TYPE I", 0, 0, 0, 0, 0, 0);    /* image-type xfer */
ftpCommand (ctrlSock, "STOR %s", file, 0, 0, 0, 0, 0); /* init file write */
```

**RETURNS**

- 1 = FTP\_PRELIM (positive preliminary)
- 2 = FTP\_COMPLETE (positive completion)
- 3 = FTP\_CONTINUE (positive intermediate)

4 = FTP\_TRANSIENT (transient negative completion)

5 = FTP\_ERROR (permanent negative completion)

ERROR if there is a read/write error or an unexpected EOF.

SEE ALSO **ftpLib**, **ftpReplyGet()**

---

## ftpCommandEnhanced()

NAME **ftpCommandEnhanced()** – send an FTP command and get the complete RFC reply code

SYNOPSIS

```
int ftpCommandEnhanced
(
    int    ctrlSock,          /* fd of control connection socket */
    char * fmt,              /* format string of command to send */
    int    arg1,             /* first of six args to format string */
    int    arg2,
    int    arg3,
    int    arg4,
    int    arg5,
    int    arg6,
    char * replyString,      /* storage for the last line of the server */
                                /* response or NULL */
    int    replyStringLength /* Maximum character length of the replyString */
)
```

DESCRIPTION This command supersedes **ftpCommand()**

This routine sends the specified command on the specified socket, which should be a control connection to a remote FTP server. The command is specified as a string in **printf()** format with up to six arguments.

After the command is sent, **ftpCommand()** waits for the reply from the remote server. The FTP reply code is returned in the same way as in **ftpReplyGetEnhanced()**.

EXAMPLE

```
ftpCommandEnhanced (ctrlSock, "TYPE I", 0, 0, 0, 0, 0, 0, 0, 0, 0);
                                /* image-type xfer */
ftpCommandEnhanced (ctrlSock, "STOR %s", file, 0, 0, 0, 0, 0, 0, 0, 0);
                                /* init file write */
ftpCommandEnhanced (ctrlSock, "PASV", file, 0, 0, 0, 0, 0, reply, rplyLen);
                                /* Get port */
```

RETURNS The complete FTP response code (see RFC #959)

ERROR if there is a read/write error or an unexpected EOF.

SEE ALSO **ftpLib**, **ftpReplyGetEnhanced()**, **ftpReplyGet()**

---

## ftpDataConnGet()

NAME **ftpDataConnGet()** – get a completed FTP data connection

SYNOPSIS

```
int ftpDataConnGet
(
    int dataSock          /* fd of data socket on which to await */
                          /* connection */
)
```

DESCRIPTION This routine completes a data connection initiated by a call to **ftpDataConnInit()**. It waits for a connection on the specified socket from the remote FTP server. The specified socket should be the one returned by **ftpDataConnInit()**. The connection is established on a new socket, whose file descriptor is returned as the result of this function. The original socket, specified in the argument to this routine, is closed.

Usually this routine is called after **ftpDataConnInit()** and **ftpCommand()** to initiate a data transfer from/to the remote FTP server.

RETURNS The file descriptor of the new data socket, or **ERROR** if the connection failed.

SEE ALSO **ftpLib**, **ftpDataConnInit()**, **ftpCommand()**

---

## ftpDataConnInit()

NAME **ftpDataConnInit()** – initialize an FTP data connection using PORT mode

SYNOPSIS

```
int ftpDataConnInit
(
    int ctrlSock          /* fd of associated control socket */
)
```

DESCRIPTION This routine sets up the client side of a data connection for the specified control connection using the PORT command. It creates the data port, informs the remote FTP server of the data port address, and listens on that data port. The server will then connect

to this data port in response to a subsequent data-transfer command sent on the control connection (see the manual entry for **ftpCommand( )**).

This routine must be called *before* the data-transfer command is sent; otherwise, the server's connect may fail.

This routine is called after **ftpHookup( )** and **ftpLogin( )** to establish a connection with a remote FTP server at the lowest level. (For a higher-level interaction with a remote FTP server, see **ftpXfer( )**.)

Please note that **ftpDataConnInitPassiveMode( )** is recommended instead of **ftpDataConnInit( )**.

**RETURNS** The file descriptor of the data socket created, or **ERROR**.

**SEE ALSO** **ftpLib**, **ftpDataConnInitPassiveMode( )**, **ftpHookup( )**, **ftpLogin( )**, **ftpCommand( )**, **ftpXfer( )**

---

## **ftpDataConnInitPassiveMode( )**

**NAME** **ftpDataConnInitPassiveMode( )** – initialize an FTP data connection using PASV mode

**SYNOPSIS**

```
int ftpDataConnInitPassiveMode
(
    int ctrlSock          /* fd of associated control socket */
)
```

**DESCRIPTION** This routine sets up the client side of a data connection for the specified control connection. It issues a PASV command and attempts to connect to the host-specified port. If the host responds that it can not process the PASV command (command not supported) or fails to recognize the command, it will return **ERROR**.

This routine must be called *before* the data-transfer command is sent; otherwise, the server's connect may fail.

This routine is called after **ftpHookup( )** and **ftpLogin( )** to establish a connection with a remote FTP server at the lowest level. (For a higher-level interaction with a remote FTP server, see **ftpXfer( )**.)

This function is preferred over **ftpDataConnInit( )** because the remote system must preserve old port connection pairs even if the target system suffers from a reboot (2MSL). Using PASV we encourage the host's selection of a fresh port.

**RETURNS** The file descriptor of the data socket created, or **ERROR**.

**SEE ALSO** **ftpLib**, **ftpHookup( )**, **ftpLogin( )**, **ftpCommandEnhanced( )**, **ftpXfer( )**, **ftpConnInit( )**

---

## ftpdDelete()

**NAME** ftpdDelete() – terminate the FTP server task

**SYNOPSIS** `STATUS ftpdDelete (void)`

**DESCRIPTION** This routine halts the FTP server and closes the control connection. All client sessions are removed after completing any commands in progress. When this routine executes, no further client connections will be accepted until the server is restarted. This routine is not reentrant and must not be called from interrupt level.

---

**NOTE:** If any file transfer operations are in progress when this routine is executed, the transfers will be aborted, possibly leaving incomplete files on the destination host.

---

**RETURNS** OK if shutdown completed, or **ERROR** otherwise.

**ERRNO** N/A

**SEE ALSO** ftpdLib

---

## ftpdInit()

**NAME** ftpdInit() – initialize the FTP server task

**SYNOPSIS** `STATUS ftpdInit  
(  
    FUNCPTR pLoginRtn,           /* user verification routine, or NULL */  
    int     stackSize           /* task stack size, or 0 for default */  
)`

**DESCRIPTION** This routine installs the password verification routine indicated by *pLoginRtn* and establishes a control connection for the primary FTP server task, which it then creates. It is called automatically during system startup if `INCLUDE_FTP_SERVER` is defined. The primary server task supports simultaneous client sessions, up to the limit specified by the global variable `ftpsMaxClients`. The default value allows a maximum of four simultaneous connections. The *stackSize* argument specifies the stack size for the primary server task. It is set to the value specified in the `ftpdWorkTaskStackSize` global variable by default.

**RETURNS** OK if server started, or **ERROR** otherwise.

**ERRNO** N/A

**SEE ALSO** ftpdLib

---

## ftpHookup()

**NAME** ftpHookup() – get a control connection to the FTP server on a specified host

**SYNOPSIS**

```
int ftpHookup
(
    char * host                /* server host name or inet address */
)
```

**DESCRIPTION** This routine establishes a control connection to the FTP server on the specified host. This is the first step in interacting with a remote FTP server at the lowest level. (For a higher-level interaction with a remote FTP server, see the manual entry for **ftpXfer()**.)

**RETURNS** The file descriptor of the control socket, or **ERROR** if the Internet address or the host name is invalid, if a socket could not be created, or if a connection could not be made.

**SEE ALSO** ftpLib, ftpLogin(), ftpXfer()

---

## ftpLibDebugOptionSet()

**NAME** ftpLibDebugOptionSet() – set the debug level of the ftp library routines

**SYNOPSIS**

```
void ftpLibDebugOptionSet
(
    UINT32 debugLevel
)
```

**DESCRIPTION** This routine enables the debugging of ftp transactions using the ftp library.

| Debugging Level     | Meaning                         |
|---------------------|---------------------------------|
| FTPL_DEBUG_OFF      | No debugging messages.          |
| FTPL_DEBUG_INCOMING | Display all incoming responses. |
| FTPL_DEBUG_OUTGOING | Display all outgoing commands.  |
| FTPL_DEBUG_ERRORS   | Display warnings and errors     |

**EXAMPLE**

```
ftpLibDebugOptionsSet (FTPL_DEBUG_ERRORS); /* Display any runtime errors */
ftpLibDebugOptionsSet (FTPL_DEBUG_OUTGOING); /* Display outgoing commands */
ftpLibDebugOptionsSet (FTPL_DEBUG_INCOMING); /* Display incoming replies */
ftpLibDebugOptionsSet (FTPL_DEBUG_INCOMING | /* Display both commands and */
                      FTPL_DEBUG_OUTGOING); /* replies */
```

**RETURNS** N/A

**SEE ALSO** ftpLib

**F**

---

## ftpLogin()

**NAME** ftpLogin() – log in to a remote FTP server

**SYNOPSIS**

```
STATUS ftpLogin
(
    int    ctrlSock,          /* fd of login control socket */
    char * user,             /* user name for host login */
    char * passwd,          /* password for host login */
    char * account           /* account for host login */
)
```

**DESCRIPTION** This routine logs in to a remote server with the specified user name, password, and account name, as required by the specific remote host. This is typically the next step after calling **ftpHookup()** in interacting with a remote FTP server at the lowest level. (For a higher-level interaction with a remote FTP server, see the manual entry for **ftpXfer()**).

**RETURNS** OK, or ERROR if the routine is unable to log in.

**SEE ALSO** ftpLib, ftpHookup(), ftpXfer()

---

## ftpLs()

**NAME** ftpLs() – list directory contents via FTP

**SYNOPSIS**

```
STATUS ftpLs
(
    char * dirName          /* name of directory to list */
)
```

|                    |                                                                                                                                                                                                                                                                           |
|--------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>DESCRIPTION</b> | This routine lists the contents of a directory. The content list is obtained via an NLST FTP transaction.<br><br>The local device name must be the same as the remote host name with a colon ":" as a suffix. (For example "wrs:" is the device name for the "wrs" host.) |
| <b>RETURNS</b>     | OK, or ERROR if could not open directory.                                                                                                                                                                                                                                 |
| <b>SEE ALSO</b>    | ftpLib                                                                                                                                                                                                                                                                    |

---

## ftpReplyGet( )

**NAME** ftpReplyGet( ) – get an FTP command reply

**SYNOPSIS**

```
int ftpReplyGet
(
    int  ctrlSock,          /* control socket fd of FTP connection */
    BOOL expecteof         /* TRUE = EOF expected, FALSE = EOF is error */
)
```

**DESCRIPTION**

This routine has been superseded by **ftpReplyGetEnhanced( )**.

This routine gets a command reply on the specified control socket.

The three-digit reply code from the first line is saved and interpreted. The left-most digit of the reply code identifies the type of code (see RETURNS below).

The caller's error status is always set to the complete three-digit reply code regardless of the actual reply value (see the manual entry for **errnoGet( )**). If the reply code indicates an error, the entire reply is printed if the ftp error printing is enabled (see the manual entry for **ftpLibDebugOptionsSet( )**).

If an EOF is encountered on the specified control socket, but no EOF was expected (*expecteof* == FALSE), then **ERROR** is returned.

**RETURNS**

- 1 = FTP\_PRELIM (positive preliminary)
- 2 = FTP\_COMPLETE (positive completion)
- 3 = FTP\_CONTINUE (positive intermediate)
- 4 = FTP\_TRANSIENT (transient negative completion)
- 5 = FTP\_ERROR (permanent negative completion)

**ERROR** if there is a read/write error or an unexpected EOF.

**SEE ALSO** ftpLib

---

## ftpReplyGetEnhanced()

**NAME** `ftpReplyGetEnhanced()` – get an FTP command reply

**SYNOPSIS**

```
int ftpReplyGetEnhanced
(
    int    ctrlSock,          /* control socket fd of FTP connection */
    BOOL   expecteof        /* TRUE = EOF expected, FALSE = EOF is error */
    char * replyString,      /* Location to store text of reply, or NULL */
    int    stringLengthMax /* Maximum length of reply (not including NULL) */
)
```

**DESCRIPTION** This routine supersedes `ftpReplyGet()`

This routine gets a command reply on the specified control socket.

The three-digit reply code from the first line is saved and interpreted. The left-most digit of the reply code identifies the type of code (see RETURNS below).

The caller's error status is always set to the complete three-digit reply code (see the manual entry for `errnoGet()`). If the reply code indicates an error, the entire reply is printed if the ftp error printing is enabled (see the manual entry for `ftpLibDebugOptionsSet()`).

The last line of text retrieved from the servers response is stored in the location specified by *replyString*. If *replyString* is NULL the parameter is ignored.

If an EOF is encountered on the specified control socket, but no EOF was expected (*expecteof* == FALSE), then **ERROR** is returned.

**RETURNS** The complete FTP response code (see RFC #959)

**ERROR** if there is a read/write error or an unexpected EOF.

**SEE ALSO** `ftpLib`

---

## ftpTransientConfigGet()

|                    |                                                                                                                                                                                                       |
|--------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>        | <code>ftpTransientConfigGet()</code> – get parameters for host FTP_TRANSIENT responses                                                                                                                |
| <b>SYNOPSIS</b>    | <pre>STATUS ftpTransientConfigGet (     UUINT32 * maxRetryCount, /* The maximum number of attempts to retry */     UUINT32 * retryInterval /* time (in system clock ticks) between retries */ )</pre> |
| <b>DESCRIPTION</b> | This routine retrieves the delay between retries in response to receiving FTP_TRANSIENT and the maximum retry count permitted before failing.                                                         |
| <b>RETURNS</b>     | OK                                                                                                                                                                                                    |
| <b>SEE ALSO</b>    | <code>ftpLib</code> , <code>ftpTransientConfigSet()</code> , <code>tickLib</code>                                                                                                                     |

---

## ftpTransientConfigSet()

|                    |                                                                                                                                                                                                   |
|--------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>        | <code>ftpTransientConfigSet()</code> – set parameters for host FTP_TRANSIENT responses                                                                                                            |
| <b>SYNOPSIS</b>    | <pre>STATUS ftpTransientConfigSet (     UUINT32 maxRetryCount, /* The maximum number of attempts to retry */     UUINT32 retryInterval /* time (in system clock ticks) between retries */ )</pre> |
| <b>DESCRIPTION</b> | This routine adjusts the delay between retries in response to receiving FTP_PRELIM and the maximum retry count permitted before failing.                                                          |
| <b>RETURNS</b>     | OK                                                                                                                                                                                                |
| <b>SEE ALSO</b>    | <code>ftpLib</code>                                                                                                                                                                               |

---

## ftpTransientFatalInstall( )

**NAME** ftpTransientFatalInstall( ) – set applette to stop FTP transient host responses

**SYNOPSIS**

```
STATUS ftpTransientFatalInstall
(
    FUNCPTR pApplette /* function that returns TRUE or FALSE */
)
```

**DESCRIPTION** The routine installs a function which will determine if a transient response should be fatal. Some FTP servers incorrectly use **transient** responses instead of **error** to describe conditions such as **disk full**.

**RETURNS** OK if the installation is successful, or **ERROR** if the installation fails.

**SEE ALSO** ftpLib, ftpTransientConfigSet( ), ftpTransientFatal( ) in target/config/comps/src/net/usrFtp.c.

---

## ftpXfer( )

**NAME** ftpXfer( ) – initiate a transfer via FTP

**SYNOPSIS**

```
STATUS ftpXfer
(
    char * host,           /* name of server host */
    char * user,          /* user name for host login */
    char * passwd,       /* password for host login */
    char * acct,         /* account for host login */
    char * cmd,          /* command to send to host */
    char * dirname,     /* directory to cd to before sending command */
    char * filename,    /* filename to send with command */
    int * pCtrlSock,    /* where to return control socket fd */
    int * pDataSock     /* where to return data socket fd, (NULL == */
                        /* don't open data connection) */
)
```

**DESCRIPTION** This routine initiates a transfer via a remote FTP server in the following order:

- (1) Establishes a connection to the FTP server on the specified host.
- (2) Logs in with the specified user name, password, and account, as necessary for the particular host.

- (3) Sets the transfer type to image by sending the command "TYPE I".
- (4) Changes to the specified directory by sending the command "CWD *dirname*".
- (5) Sends the specified transfer command with the specified filename as an argument, and establishes a data connection. Typical transfer commands are "STOR %s", to write to a remote file, or "RETR %s", to read a remote file.

The resulting control and data connection file descriptors are returned via *pCtrlSock* and *pDataSock*, respectively.

After calling this routine, the data can be read or written to the remote server by reading or writing on the file descriptor returned in *pDataSock*. When all incoming data has been read (as indicated by an EOF when reading the data socket) and/or all outgoing data has been written, the data socket fd should be closed. The routine **ftpReplyGet()** should then be called to receive the final reply on the control socket, after which the control socket should be closed.

If the FTP command does not involve data transfer, *pDataSock* should be `NULL`, in which case no data connection will be established. The only FTP commands supported for this case are DELE, RMD, and MKD.

**EXAMPLE**

The following code fragment reads the file `/usr/fred/myfile` from the host "server", logged in as user "fred", with password "magic" and no account name.

```
#include "vxWorks.h"
#include "ftpLib.h"
int      ctrlSock;
int      dataSock;
char     buf [512];
int      nBytes;
STATUS   status;
if (ftpXfer ("server", "fred", "magic", "",
            "RETR %s", "/usr/fred", "myfile",
            &ctrlSock, &dataSock) == ERROR)
    return (ERROR);
while ((nBytes = read (dataSock, buf, sizeof (buf))) > 0)
{
    ...
}
close (dataSock);
if (nBytes < 0)          /* read error? */
    status = ERROR;
if (ftpReplyGet (ctrlSock, TRUE) != FTP_COMPLETE)
    status = ERROR;
if (ftpCommand (ctrlSock, "QUIT", 0, 0, 0, 0, 0, 0) != FTP_COMPLETE)
    status = ERROR;
close (ctrlSock);
```

**RETURNS** OK, or **ERROR** if any socket cannot be created or if a connection cannot be made.

**SEE ALSO** `ftpLib`, `ftpReplyGet()`

---

## ftruncate( )

**NAME** `ftruncate()` – truncate a file (POSIX)

**SYNOPSIS**

```
int ftruncate
(
    int  fildes,          /* fd of file to truncate */
    off_t length         /* length to truncate file */
)
```

**DESCRIPTION** This routine truncates a file to a specified size.

**RETURNS** 0 (OK) or -1 (**ERROR**) if unable to truncate file.

**ERRNO** EROFS - File resides on a read-only file system.  
EBADF - File is open for reading only.  
EINVAL - File descriptor refers to a file on which this operation is impossible.

**SEE ALSO** `ftruncate`

---

## fwrite( )

**NAME** `fwrite()` – write from a specified array (ANSI)

**SYNOPSIS**

```
int fwrite
(
    const void * buf,     /* where to copy from */
    size_t     size,     /* element size */
    size_t     count,    /* no. of elements */
    FILE *     fp        /* stream to write to */
)
```

**DESCRIPTION** This routine writes, from the array *buf*, up to *count* elements whose size is *size*, to a specified stream. The file position indicator for the stream (if defined) is advanced by the

***fwrite()***

number of characters successfully written. If an error occurs, the resulting value of the file position indicator for the stream is indeterminate.

**INCLUDE FILES**     **stdio.h**

**RETURNS**           The number of elements successfully written, which will be less than *count* only if a write error is encountered.

**SEE ALSO**           **ansiStdio**

---

## getc()

**NAME** `getc()` – return the next character from a stream (ANSI)

**SYNOPSIS**

```
int getc
(
    FILE * fp                /* input stream */
)
```

**DESCRIPTION** This routine is equivalent to `fgetc()`, except that if it is implemented as a macro, it may evaluate `fp` more than once; thus the argument should never be an expression with side effects.

If the stream is at end-of-file, the end-of-file indicator for the stream is set; if a read error occurs, the error indicator is set.

**INCLUDE FILES** `stdio.h`

**RETURNS** The next character from the stream, or EOF if the stream is at end-of-file or a read error occurs.

**SEE ALSO** `ansiStdio`, `fgetc()`

---

## getchar()

**NAME** `getchar()` – return the next character from the standard input stream (ANSI)

**SYNOPSIS** `int getchar (void)`

**DESCRIPTION** This routine returns the next character from the standard input stream and advances the file position indicator.

It is equivalent to `getc()` with the stream argument `stdin`.

If the stream is at end-of-file, the end-of-file indicator is set; if a read error occurs, the error indicator is set.

**INCLUDE FILES** `stdio.h`

**RETURNS** The next character from the standard input stream, or EOF if the stream is at end-of-file or a read error occurs.

**SEE ALSO** `ansiStdio`, `getc()`, `fgetc()`

**getcwd()**

---

**getcwd()****NAME** getcwd() – get the current default path (POSIX)

**SYNOPSIS**

```
char *getcwd
(
    char * buffer,          /* where to return the pathname */
    int   size             /* size in bytes of buffer */
)
```

**DESCRIPTION** This routine copies the name of the current default path to *buffer*. It provides the same functionality as **ioDefPathGet()** and is provided for POSIX compatibility.**RETURNS** A pointer to the supplied buffer, or NULL if *size* is too small to hold the current default path.**SEE ALSO** **ioLib**, **ioDefPathSet()**, **ioDefPathGet()**, **chdir()**

---

**getenv()****NAME** getenv() – get an environment variable (ANSI)

**SYNOPSIS**

```
char *getenv
(
    const char * name      /* env variable to get value for */
)
```

**DESCRIPTION** This routine searches the environment list (see the UNIX BSD 4.3 manual entry for **environ(5V)**) for a string of the form “name=value” and returns the value portion of the string, if the string is present; otherwise it returns a NULL pointer.**RETURNS** A pointer to the string value, or a NULL pointer.**SEE ALSO** **envLib**, **envLibInit()**, **putenv()**, UNIX BSD 4.3 manual entry for **environ(5V)**, *American National Standard for Information Systems -Programming Language - C, ANSI X3.159-1989: General Utilities (stdlib.h)*

---

## gethostname()

**NAME** `gethostname()` – get the symbolic name of this machine

**SYNOPSIS**

```
int gethostname
(
    char * name,          /* machine name */
    int  nameLen         /* length of name */
)
```

**DESCRIPTION** This routine gets the target machine's symbolic name, which can be used for identification.

**RETURNS** OK or ERROR.

**SEE ALSO** `hostLib`

---

## getpeername()

**NAME** `getpeername()` – get the name of a connected peer

**SYNOPSIS**

```
STATUS getpeername
(
    int          s,      /* socket descriptor */
    struct sockaddr * name, /* where to put name */
    int *        namelen /* space available in name, later filled in */
                        /* with actual name size */
)
```

**DESCRIPTION** This routine gets the name of the peer connected to socket *s*. The *namelen* parameter should be initialized to indicate the amount of space referenced by *name*. On return, the name of the socket is copied to *name* and the actual size of the socket name is copied to *namelen*.

**RETURNS** OK, or ERROR if the socket is invalid or not connected.

**SEE ALSO** `sockLib`

**gets()**

---

**gets()****NAME** gets() – read characters from the standard input stream (ANSI)**SYNOPSIS**  

```
char * gets
(
    char * buf          /* output array */
)
```

**DESCRIPTION** This routine reads characters from the standard input stream into the array *buf* until end-of-file is encountered or a new-line is read. Any new-line character is discarded, and a null character is written immediately after the last character read into the array.

If end-of-file is encountered and no characters have been read, the contents of the array remain unchanged. If a read error occurs, the array contents are indeterminate.

**INCLUDE FILES** `stdio.h`**RETURNS** A pointer to *buf*, or a null pointer if (1) end-of-file is encountered and no characters have been read, or (2) there is a read error.**SEE ALSO** `ansiStdio`

---

**getsockname()****NAME** getsockname() – get a socket name**SYNOPSIS**  

```
STATUS getsockname
(
    int          s,          /* socket descriptor */
    struct sockaddr * name, /* where to return name */
    int *       namelen /* space available in name, later filled in */
                    /* with actual name size */
)
```

**DESCRIPTION** This routine gets the current name for the specified socket *s*. The *namelen* parameter should be initialized to indicate the amount of space referenced by *name*. On return, the name of the socket is copied to *name* and the actual size of the socket name is copied to *namelen*.**RETURNS** OK, or ERROR if the socket is invalid.

SEE ALSO `sockLib`

---

## getsockopt()

NAME `getsockopt()` – get socket options

SYNOPSIS

```
STATUS getsockopt
(
    int    s,                /* socket */
    int    level,            /* protocol level for options */
    int    optname,          /* name of option */
    char * optval,           /* where to put option */
    int *  optlen            /* where to put option length */
)
```

DESCRIPTION

This routine returns relevant option values associated with a socket. To manipulate options at the “socket” level, *level* should be `SOL_SOCKET`. Any other levels should use the appropriate protocol number. The *optlen* parameter should be initialized to indicate the amount of space referenced by *optval*. On return, the value of the option is copied to *optval* and the actual size of the option is copied to *optlen*.

Although *optval* is passed as a `char *`, the actual variable whose address gets passed in should be an integer or a structure, depending on which *optname* is being passed. Refer to `setsockopt()` to determine the correct type of the actual variable (whose address should then be cast to a `char *`).

RETURNS

OK, or `ERROR` if there is an invalid socket, an unknown option, or the call is unable to get the specified option.

EXAMPLE

Because `SO_REUSEADDR` has an integer parameter, the variable to be passed to `getsockopt()` should be declared as

```
int reuseVal;
```

and passed in as

```
(char *)&reuseVal.
```

Otherwise the user might mistakenly declare `reuseVal` as a character, in which case `getsockopt()` will only return the first byte of the integer representing the state of this option. Then whether the return value is correct or always 0 depends on the endianness of the machine.

SEE ALSO `sockLib`, `setsockopt()`

**getw()**

---

**getw()****NAME** `getw()` – read the next word (32-bit integer) from a stream**SYNOPSIS**  

```
int getw
(
    FILE * fp                /* stream to read from */
)
```

**DESCRIPTION**  
This routine reads the next 32-bit quantity from a specified stream. It returns EOF on end-of-file or an error; however, this is also a valid integer, thus **feof()** and **ferror()** must be used to check for a true end-of-file.

This routine is provided for compatibility with earlier VxWorks releases.

**INCLUDE FILES** `stdio.h`**RETURN** A 32-bit number from the stream, or EOF on either end-of-file or an error.**SEE ALSO** `ansiStdio`, `putw()`

---

**getwd()****NAME** `getwd()` – get the current default path**SYNOPSIS**  

```
char *getwd
(
    char * pathname         /* where to return the pathname */
)
```

**DESCRIPTION**  
This routine copies the name of the current default path to *pathname*. It provides the same functionality as `ioDefPathGet()` and `getcwd()`. It is provided for compatibility with some older UNIX systems.

The parameter *pathname* should be `MAX_FILENAME_LENGTH` characters long.

**RETURNS** A pointer to the resulting path name.**SEE ALSO** `ioLib`

---

## gmtime()

|                      |                                                                                                                                                                                                                                       |
|----------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>          | <b>gmtime()</b> – convert calendar time into UTC broken-down time (ANSI)                                                                                                                                                              |
| <b>SYNOPSIS</b>      | <pre>struct tm *gmtime (     const time_t * timer      /* calendar time in seconds */ )</pre>                                                                                                                                         |
| <b>DESCRIPTION</b>   | <p>This routine converts the calendar time pointed to by <i>timer</i> into broken-down time, expressed as Coordinated Universal Time (UTC).</p> <p>This routine is not reentrant. For a reentrant version, see <b>gmtime_r()</b>.</p> |
| <b>INCLUDE FILES</b> | <b>time.h</b>                                                                                                                                                                                                                         |
| <b>RETURNS</b>       | A pointer to a broken-down time structure ( <b>tm</b> ), or a null pointer if UTC is not available.                                                                                                                                   |
| <b>SEE ALSO</b>      | <b>ansiTime</b>                                                                                                                                                                                                                       |

---

## gmtime\_r()

|                      |                                                                                                                                                                                                                                                                              |
|----------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>          | <b>gmtime_r()</b> – convert calendar time into broken-down time (POSIX)                                                                                                                                                                                                      |
| <b>SYNOPSIS</b>      | <pre>int gmtime_r (     const time_t * timer,      /* calendar time in seconds */     struct tm *   timeBuffer /* buffer for broken down time */ )</pre>                                                                                                                     |
| <b>DESCRIPTION</b>   | <p>This routine converts the calendar time pointed to by <i>timer</i> into broken-down time, expressed as Coordinated Universal Time (UTC). The broken-down time is stored in <i>timeBuffer</i>.</p> <p>This routine is the POSIX re-entrant version of <b>gmtime()</b>.</p> |
| <b>INCLUDE FILES</b> | <b>time.h</b>                                                                                                                                                                                                                                                                |
| <b>RETURNS</b>       | OK.                                                                                                                                                                                                                                                                          |
| <b>SEE ALSO</b>      | <b>ansiTime</b>                                                                                                                                                                                                                                                              |



---

## h()

|                    |                                                                                                                                                                                                                                                                |
|--------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>        | <b>h()</b> – display or set the size of shell history                                                                                                                                                                                                          |
| <b>SYNOPSIS</b>    | <pre>void h (     int size          /* 0 = display, &gt;0 = set history to new size */ ) </pre>                                                                                                                                                                |
| <b>DESCRIPTION</b> | This command displays or sets the size of VxWorks shell history. If no argument is specified, shell history is displayed. If <i>size</i> is specified, that number of the most recent commands is saved for display. The value of <i>size</i> is initially 20. |
| <b>RETURNS</b>     | N/A                                                                                                                                                                                                                                                            |
| <b>SEE ALSO</b>    | <b>usrLib</b> , <b>shellHistory()</b> , <b>ledLib</b> , <i>VxWorks Programmer's Guide: Target Shell</i> , <b>windsh</b> , <i>Tornado User's Guide: Shell</i>                                                                                                   |

---

## hashFuncIterScale()

|                    |                                                                                                                                                                                                                                                                                                           |
|--------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>        | <b>hashFuncIterScale()</b> – iterative scaling hashing function for strings                                                                                                                                                                                                                               |
| <b>SYNOPSIS</b>    | <pre>int hashFuncIterScale (     int          elements, /* number of elements in hash table */     H_NODE_STRING * pNode, /* pointer to string keyed hash node */     int          seed      /* seed to be used as scalar */ ) </pre>                                                                     |
| <b>DESCRIPTION</b> | This hashing function interprets the key as a pointer to a null terminated string. A seed of 13 or 27 appears to work well. It calculates the hash as follows: <pre>for (tkey = pNode-&gt;string; *tkey != '\0'; tkey++)     hash = hash * seed + (unsigned int) *tkey; hash &amp;= (elements - 1);</pre> |
| <b>RETURNS</b>     | integer between 0 and (elements - 1)                                                                                                                                                                                                                                                                      |
| <b>SEE ALSO</b>    | <b>hashLib</b>                                                                                                                                                                                                                                                                                            |

---

## hashFuncModulo()

**NAME** hashFuncModulo() – hashing function using remainder technique

**SYNOPSIS**

```
int hashFuncModulo
(
    int          elements,      /* number of elements in hash table */
    H_NODE_INT * pNode,        /* pointer to integer keyed hash node */
    int          divisor        /* divisor */
)
```

**DESCRIPTION** This hashing function interprets the key as a 32 bit quantity and applies the standard hashing function:  $h(k) = K \text{ mod } D$ . Where D is the passed divisor. The result of the hash function is masked to the appropriate number of bits to ensure the hash is not greater than (elements - 1).

**RETURNS** integer between 0 and (elements - 1)

**SEE ALSO** hashLib

---

## hashFuncMultiply()

**NAME** hashFuncMultiply() – multiplicative hashing function

**SYNOPSIS**

```
int hashFuncMultiply
(
    int          elements,      /* number of elements in hash table */
    H_NODE_INT * pNode,        /* pointer to integer keyed hash node */
    int          multiplier     /* multiplier */
)
```

**DESCRIPTION** This hashing function interprets the key as a unsigned integer quantity and applies the standard hashing function:  $h(k) = \text{leading } N \text{ bits of } (B * K)$ . Where N is the appropriate number of bits such that the hash is not greater than (elements - 1). The overflow of  $B * K$  is discarded. The value of B is passed as an argument. The choice of B is similar to that of the seed to a linear congruential random number generator. Namely, B's value should take on a large number (roughly 9 digits base 10) and end in ...x21 where x is an even number. (Don't ask... it involves statistics mumbo jumbo)

**RETURNS** integer between 0 and (elements - 1)

**SEE ALSO** hashLib

---

## hashKeyCmp()

**NAME** hashKeyCmp() – compare keys as 32 bit identifiers

**SYNOPSIS**

```
BOOL hashKeyCmp
(
    H_NODE_INT * pMatchHNode, /* hash node to match */
    H_NODE_INT * pNode,       /* hash node in table to compare to */
    int         keyCmpArg     /* argument ingnored */
)
```

**DESCRIPTION** This routine compares hash node keys as 32 bit identifiers. The argument keyCmpArg is unneeded by this comparator.

**RETURNS** TRUE if keys match or, FALSE if keys do not match.

**SEE ALSO** hashLib

---

## hashKeyStrCmp()

**NAME** hashKeyStrCmp() – compare keys based on strings they point to

**SYNOPSIS**

```
BOOL hashKeyStrCmp
(
    H_NODE_STRING * pMatchHNode, /* hash node to match */
    H_NODE_STRING * pNode,       /* hash node in table to compare to */
    int           keyCmpArg     /* argument ingnored */
)
```

**DESCRIPTION** This routine compares keys based on the strings they point to. The strings must be null terminated. The routine **strcmp()** is used to compare keys. The argument keyCmpArg is unneeded by this comparator.

**RETURNS** TRUE if keys match or, FALSE if keys do not match.

**SEE ALSO** hashLib

---

## hashLibInit()

|                    |                                                  |
|--------------------|--------------------------------------------------|
| <b>NAME</b>        | hashLibInit() – initialize hash table library    |
| <b>SYNOPSIS</b>    | <b>STATUS</b> hashLibInit (void)                 |
| <b>DESCRIPTION</b> | This routine initializes the hash table package. |
| <b>SEE ALSO</b>    | hashLib                                          |

---

## hashTblCreate()

|                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
|--------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>        | hashTblCreate() – create a hash table                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
| <b>SYNOPSIS</b>    | <pre><b>HASH_ID</b> hashTblCreate (     int      sizeLog2,          /* number of elements in hash table log 2 */     FUNCPTR  keyCmpRtn,        /* function to test keys for equivalence */     FUNCPTR  keyRtn,           /* hashing function to generate hash from key */     int      keyArg            /* argument to hashing function */ )</pre>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
| <b>DESCRIPTION</b> | <p>This routine creates a hash table <math>2^{\text{sizeLog2}}</math> number of elements. The hash table is carved from the system memory pool via <b>malloc (2)</b>. To accommodate the list structures associated with the table, the actual amount of memory allocated will be roughly eight times the number of elements requested. Additionally, two routines must be specified to dictate the behavior of the hashing table. The first routine is the hashing function.</p> <p>The hashing function's role is to disperse the hash nodes added to the table as evenly throughout the table as possible. The hashing function receives as its parameters; the number of elements in the table, a pointer to the <b>HASH_NODE</b> structure, and finally the <i>keyArg</i> parameter passed to this routine. The <i>keyArg</i> may be used to seed the hashing function. The hash function returns an index between 0 and (elements - 1). Standard hashing functions are available in this library.</p> <p>The <i>keyCmpRtn</i> parameter specifies the other function required by the hash table. This routine tests for equivalence of two <b>HASH_NODES</b>. It returns a boolean, <b>TRUE</b> if the keys match, and <b>FALSE</b> if they differ. As an example, a hash node may contain a <b>HASH_NODE</b> followed by a key which is an unsigned integer identifiers, or a pointer to a string, depending on the application. Standard hash node comparators are available in this library.</p> |

**RETURNS** HASH\_ID, or NULL if hash table could not be created.

**SEE ALSO** hashLib, hashFuncIterScale(), hashFuncModulo(), hashFuncMultiply(), hashKeyCmp(), hashKeyStrCmp()

---

## hashTblDelete()

**NAME** hashTblDelete() – delete a hash table

**SYNOPSIS**

```
STATUS hashTblDelete
(
    HASH_ID hashId          /* id of hash table to delete */
)
```

**DESCRIPTION** This routine deletes the specified hash table and frees the associated memory. The hash table is marked as invalid.

**RETURNS** OK, or ERROR if hashId is invalid.

**SEE ALSO** hashLib

---

## hashTblDestroy()

**NAME** hashTblDestroy() – destroy a hash table

**SYNOPSIS**

```
STATUS hashTblDestroy
(
    HASH_ID hashId,          /* id of hash table to destroy */
    BOOL    dealloc         /* deallocate associated memory */
)
```

**DESCRIPTION** This routine destroys the specified hash table and optionally frees the associated memory. The hash table is marked as invalid.

**RETURNS** OK, or ERROR if hashId is invalid.

**SEE ALSO** hashLib

---

## hashTblEach()

**NAME** hashTblEach() – call a routine for each node in a hash table

**SYNOPSIS**

```
HASH_NODE *hashTblEach
(
    HASH_ID hashId,          /* hash table to call routine for */
    FUNCPTR routine,        /* the routine to call for each hash node */
    int routineArg          /* arbitrary user-supplied argument */
)
```

**DESCRIPTION** This routine calls a user-supplied routine once for each node in the hash table. The routine should be declared as follows:

```
BOOL routine (pNode, arg)
    HASH_NODE *pNode;      /* pointer to a hash table node */
    int arg;               /* arbitrary user-supplied argument */
```

The user-supplied routine should return **TRUE** if **hashTblEach()** is to continue calling it with the remaining nodes, or **FALSE** if it is done and **hashTblEach()** can exit.

**RETURNS** NULL if traversed whole hash table, or pointer to **HASH\_NODE** that **hashTblEach()** ended with.

**SEE ALSO** hashLib

---

## hashTblFind()

**NAME** hashTblFind() – find a hash node that matches the specified key

**SYNOPSIS**

```
HASH_NODE *hashTblFind
(
    HASH_ID hashId,          /* id of hash table from which to find node */
    HASH_NODE * pMatchNode, /* pointer to hash node to match */
    int keyCmpArg           /* parameter to be passed to key comparator */
)
```

**DESCRIPTION** This routine finds the hash node that matches the specified key.

**RETURNS** pointer to **HASH\_NODE**, or NULL if no matching hash node is found.

**SEE ALSO** hashLib

---

## hashTblInit()

**NAME** hashTblInit() – initialize a hash table

**SYNOPSIS**

```

STATUS hashTblInit
(
    HASH_TBL * pHashTbl,      /* pointer to hash table to initialize */
    SL_LIST * pTblMem,       /* pointer to memory of sizeLog2 SL_LISTs */
    int      sizeLog2,       /* number of elements in hash table log 2 */
    FUNCPTR  keyCmpRtn,      /* function to test keys for equivalence */
    FUNCPTR  keyRtn,         /* hashing function to generate hash from key */
    int      keyArg          /* argument to hashing function */
)

```

**DESCRIPTION** This routine initializes a hash table.

**RETURNS** OK

**SEE ALSO** hashLib

---

## hashTblPut()

**NAME** hashTblPut() – put a hash node into the specified hash table

**SYNOPSIS**

```

STATUS hashTblPut
(
    HASH_ID   hashId,        /* id of hash table in which to put node */
    HASH_NODE * pHashNode    /* pointer to hash node to put in hash table */
)

```

**DESCRIPTION** This routine puts the specified hash node in the specified hash table. Identical nodes will be kept in FIFO order in the hash table.

**RETURNS** OK, or ERROR if hashId is invalid.

**SEE ALSO** hashLib, hashTblRemove()

---

## hashTblRemove()

**NAME** hashTblRemove() – remove a hash node from a hash table

**SYNOPSIS**

```
STATUS hashTblRemove
(
    HASH_ID    hashId,          /* id of hash table to remove node from */
    HASH_NODE * pHashNode     /* pointer to hash node to remove */
)
```

**DESCRIPTION** This routine removes the hash node that matches the specified key.

**RETURNS** OK, or ERROR if hashId is invalid or no matching hash node is found.

**SEE ALSO** hashLib

---

## hashTblTerminate()

**NAME** hashTblTerminate() – terminate a hash table

**SYNOPSIS**

```
STATUS hashTblTerminate
(
    HASH_ID hashId             /* id of hash table to terminate */
)
```

**DESCRIPTION** This routine terminates the specified hash table. The hash table is marked as invalid.

**RETURNS** OK, or ERROR if hashId is invalid.

**SEE ALSO** hashLib

## help()

**NAME** help() – print a synopsis of selected routines

**SYNOPSIS** void help (void)

**DESCRIPTION** This command prints the following list of the calling sequences for commonly used routines, mostly contained in `usrLib`.

|            |                              |                                                                                              |
|------------|------------------------------|----------------------------------------------------------------------------------------------|
| help       |                              | Print this list                                                                              |
| ioHelp     |                              | Print I/O utilities help info                                                                |
| dbgHelp    |                              | Print debug help info                                                                        |
| nfsHelp    |                              | Print nfs help info                                                                          |
| netHelp    |                              | Print network help info                                                                      |
| spyHelp    |                              | Print task histogrammer help info                                                            |
| timexHelp  |                              | Print execution timer help info                                                              |
| h          | [n]                          | Print (or set) shell history                                                                 |
| i          | [task]                       | Summary of tasks' TCBS                                                                       |
| ti         | task                         | Complete info on TCB for task                                                                |
| sp         | adr,args...                  | Spawn a task, pri=100, opt=0x19, stk=20000                                                   |
| taskSpawn  | name,pri,opt,stk,adr,args... | Spawn a task                                                                                 |
| td         | task                         | Delete a task                                                                                |
| ts         | task                         | Suspend a task                                                                               |
| tr         | task                         | Resume a task                                                                                |
| d          | [adr[,nunits[,width]]]       | Display memory                                                                               |
| m          | adr[,width]                  | Modify memory                                                                                |
| mRegs      | [reg[,task]]                 | Modify a task's registers interactively                                                      |
| pc         | [task]                       | Return task's program counter                                                                |
| version    |                              | Print VxWorks version info, and boot line                                                    |
| iam        | "user"["passwd"]             | Set user name and passwd                                                                     |
| whoami     |                              | Print user name                                                                              |
| devs       |                              | List devices                                                                                 |
| ld         | [syms[,noAbort][,"name"]]    | Load std in into memory<br>(syms = add symbols to table:<br>-1 = none, 0 = globals, 1 = all) |
| lkup       | ["substr"]                   | List symbols in system symbol table                                                          |
| lkAddr     | address                      | List symbol table entries near address                                                       |
| checkStack | [task]                       | List task stack sizes and usage                                                              |
| printErrno | value                        | Print the name of a status value                                                             |
| period     | secs,adr,args...             | Spawn task to call function periodically                                                     |
| repeat     | n,adr,args...                | Spawn task to call function n times<br>(0=forever)                                           |

**NOTE:** Arguments specifying <task> can be either task ID or name.

**RETURNS** N/A

**SEE ALSO** `usrLib`, *VxWorks Programmer's Guide: Target Shell*, `windsh`, *Tornado User's Guide: Shell*

---

## hostAdd()

**NAME** `hostAdd()` – add a host to the host table

**SYNOPSIS**

```
STATUS hostAdd
(
    char * hostName,          /* host name */
    char * hostAddr          /* host addr in standard Internet format */
)
```

**DESCRIPTION** This routine adds a host name to the local host table. This must be called before sockets on the remote host are opened, or before files on the remote host are accessed via **netDrv** or **nfsDrv**.

The host table has one entry per Internet address. More than one name may be used for an address. Additional host names are added as aliases.

**EXAMPLE**

```
-> hostAdd "wrs", "90.2"
-> hostShow
hostname      inet address      aliases
-----      -
localhost    127.0.0.1
yuba         90.0.0.3
wrs          90.0.0.2
value = 12288 = 0x3000 = _bzero + 0x18
```

**RETURNS** **OK**, or **ERROR** if the host table is full, the host name/inet address pair is already entered, the Internet address is invalid, or memory is insufficient.

**SEE ALSO** `hostLib`, `netDrv`, `nfsDrv`

---

## hostDelete()

**NAME** `hostDelete()` – delete a host from the host table

**SYNOPSIS**

```
STATUS hostDelete
(
    char * name,              /* host name or alias */
    char * addr              /* host addr in standard Internet format */
)
```

|                    |                                                                                                                                                                                  |
|--------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>DESCRIPTION</b> | This routine deletes a host name from the local host table. If <i>name</i> is a host name, the host entry is deleted. If <i>name</i> is a host name alias, the alias is deleted. |
| <b>RETURNS</b>     | OK, or ERROR if the parameters are invalid or the host is unknown.                                                                                                               |
| <b>ERRNO</b>       | S_hostLib_INVALID_PARAMETER, S_hostLib_UNKNOWN_HOST                                                                                                                              |
| <b>SEE ALSO</b>    | hostLib                                                                                                                                                                          |

---

## hostGetByAddr()

**H**

**NAME** hostGetByAddr() – look up a host in the host table by its Internet address

**SYNOPSIS**

```
STATUS hostGetByAddr
(
    int    addr,           /* inet address of host */
    char * name           /* buffer to hold name */
)
```

**DESCRIPTION** This routine finds the host name by its Internet address and copies it to *name*. The buffer *name* should be pre-allocated with (MAXHOSTNAMELEN + 1) bytes of memory and is NULL-terminated unless insufficient space is provided. If the DNS resolver library **resolvLib** has been configured in the vxWorks image, a query for the host name is sent to the DNS server, if the name was not found in the local host table.

---

**WARNING:** This routine does not look for aliases. Host names are limited to MAXHOSTNAMELEN (from **hostLib.h**) characters.

---

**RETURNS** OK, or ERROR if buffer is invalid or the host is unknown.

**SEE ALSO** hostLib, hostGetByName()

## hostGetByName()

|                    |                                                                                                                                                                                                                                                                                                                      |
|--------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>        | <b>hostGetByName()</b> – look up a host in the host table by its name                                                                                                                                                                                                                                                |
| <b>SYNOPSIS</b>    | <pre>int hostGetByName (     char * name           /* name of host */ )</pre>                                                                                                                                                                                                                                        |
| <b>DESCRIPTION</b> | This routine returns the Internet address of a host that has been added to the host table by <b>hostAdd()</b> . If the DNS resolver library <b>resolvLib</b> has been configured in the vxWorks image, a query for the host IP address is sent to the DNS server, if the name was not found in the local host table. |
| <b>RETURNS</b>     | The Internet address (as an integer), or <b>ERROR</b> if the host is unknown.                                                                                                                                                                                                                                        |
| <b>ERRNO</b>       | <b>S_hostLib_INVALID_PARAMETER</b> , <b>S_hostLib_UNKNOWN_HOST</b>                                                                                                                                                                                                                                                   |
| <b>SEE ALSO</b>    | <b>hostLib</b>                                                                                                                                                                                                                                                                                                       |

---

## hostShow()

|                    |                                                                                              |
|--------------------|----------------------------------------------------------------------------------------------|
| <b>NAME</b>        | <b>hostShow()</b> – display the host table                                                   |
| <b>SYNOPSIS</b>    | <pre>void hostShow (void)</pre>                                                              |
| <b>DESCRIPTION</b> | This routine prints a list of remote hosts, along with their Internet addresses and aliases. |
| <b>RETURNS</b>     | N/A                                                                                          |
| <b>SEE ALSO</b>    | <b>netShow</b> , <b>hostAdd()</b>                                                            |

---

## hostTblInit()

|                    |                                                                                                                                                                                                                                      |
|--------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>        | <b>hostTblInit()</b> – initialize the network host table                                                                                                                                                                             |
| <b>SYNOPSIS</b>    | <code>void hostTblInit (void)</code>                                                                                                                                                                                                 |
| <b>DESCRIPTION</b> | This routine initializes the host list data structure used by routines throughout this module. It should be called before any other routines in this module. This is done automatically if <code>INCLUDE_HOST_TBL</code> is defined. |
| <b>RETURNS</b>     | N/A                                                                                                                                                                                                                                  |
| <b>SEE ALSO</b>    | <code>hostLib</code> , <code>usrConfig</code>                                                                                                                                                                                        |

**i()****i()****NAME** i() – print a summary of each task’s TCB

**SYNOPSIS**

```
void i
(
    int taskNameOrId      /* task name or task ID, 0 = summarize all */
)
```

**DESCRIPTION** This command displays a synopsis of all the tasks in the system. The **ti()** routine provides more complete information on a specific task.

Both **i()** and **ti()** use **taskShow()**; see the documentation for **taskShow()** for a description of the output format.

**EXAMPLE**

```
-> i
  NAME          ENTRY          TID    PRI   STATUS    PC          SP          ERRNO DELAY
-----
tExcTask    _excTask    20fcb00  0  PEND     200c5fc  20fca6c    0      0
tLogTask    _logTask    20fb5b8  0  PEND     200c5fc  20fb520    0      0
tShell      _shell      20efcac  1  READY    201dc90  20ef980    0      0
tRlogind    _rlogind    20f3f90  2  PEND     2038614  20f3db0    0      0
tTelnetd    _telnetd    20f2124  2  PEND     2038614  20f2070    0      0
tNetTask    _netTask    20f7398  50  PEND     2038614  20f7340    0      0
value = 57 = 0x39 = '9'
```

**WARNING:** This command should be used only as a debugging aid, since the information is obsolete by the time it is displayed.

**RETURNS** N/A

**SEE ALSO** **usrLib**, **ti()**, **taskShow()**, *VxWorks Programmer’s Guide: Target Shell*, **windsh**, *Tornado User’s Guide: Shell*

---

## iam()

**NAME** iam() – set the remote user name and password

**SYNOPSIS**

```
STATUS iam
(
    char * newUser,          /* user name to use on remote */
    char * newPasswd       /* password to use on remote (NULL = none) */
)
```

**DESCRIPTION** This routine specifies the user name that will have access privileges on the remote machine. The user name must exist in the remote machine's */etc/passwd*, and if it has been assigned a password, the password must be specified in *newPasswd*.

Either parameter can be **NULL**, and the corresponding item will not be set.

The maximum length of the user name and the password is **MAX\_IDENTITY\_LEN**(defined in **remLib.h**).

---

**NOTE:** This routine is a more convenient version of **remCurIdSet()** and is intended to be used from the shell.

---

**RETURNS** OK, or **ERROR** if the call fails.

**SEE ALSO** **remLib**, **whoami()**, **remCurIdGet()**, **remCurIdSet()**

---

## icmpShowInit()

**NAME** icmpShowInit() – initialize ICMP show routines

**SYNOPSIS** void icmpShowInit (void)

**DESCRIPTION** This routine links the ICMP show facility into the VxWorks system. These routines are included automatically if **INCLUDE\_NET\_SHOW** and **INCLUDE\_ICMP** are defined.

**RETURNS** N/A

**SEE ALSO** **icmpShow**

---

## icmpstatShow()

|                    |                                                                                             |
|--------------------|---------------------------------------------------------------------------------------------|
| <b>NAME</b>        | <b>icmpstatShow()</b> – display statistics for ICMP                                         |
| <b>SYNOPSIS</b>    | <code>void icmpstatShow (void)</code>                                                       |
| <b>DESCRIPTION</b> | This routine displays statistics for the ICMP (Internet Control Message Protocol) protocol. |
| <b>RETURNS</b>     | N/A                                                                                         |
| <b>SEE ALSO</b>    | <code>icmpShow</code>                                                                       |

---

## ifAddrAdd()

|                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
|--------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>        | <b>ifAddrAdd()</b> – add an interface address for a network interface                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
| <b>SYNOPSIS</b>    | <pre><b>STATUS</b> ifAddrAdd (     char * interfaceName,      /* name of interface to configure */     char * interfaceAddress,  /* Internet address to assign to interface */     char * broadcastAddress,  /* broadcast address to assign to interface */     int   subnetMask         /* subnetMask */ )</pre>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| <b>DESCRIPTION</b> | <p>This routine assigns an Internet address to a specified network interface. The Internet address can be a host name or a standard Internet address format (<i>e.g.</i>, 90.0.0.4). If a host name is specified, it should already have been added to the host table with <code>hostAdd()</code>.</p> <p>You must specify both an <i>interfaceName</i> and an <i>interfaceAddress</i>. A <i>broadcastAddress</i> is optional. If <i>broadcastAddress</i> is NULL, <code>in_ifinit()</code> generates a <i>broadcastAddress</i> value based on the <i>interfaceAddress</i> value and the netmask. A <i>subnetMask</i> value is optional. If <i>subnetMask</i> is 0, <code>in_ifinit()</code> uses a <i>subnetMask</i> the same as the netmask that is generated by the <i>interfaceAddress</i>. The <i>broadcastAddress</i> is also <i>destAddress</i> in case of <code>IFF_POINTOPOINT</code>.</p> |
| <b>RETURNS</b>     | OK, or ERROR if the interface cannot be set.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| <b>SEE ALSO</b>    | <code>ifLib</code> , <code>ifAddrGet()</code> , <code>ifDstAddrSet()</code> , <code>ifDstAddrGet()</code>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |

---

## ifAddrDelete()

- NAME** `ifAddrDelete()` – delete an interface address for a network interface
- SYNOPSIS**
- ```
STATUS ifAddrDelete
(
    char * interfaceName,      /* name of interface to delete addr from */
    char * interfaceAddress   /* Internet address to delete from interface */
)
```
- DESCRIPTION** This routine deletes an Internet address from a specified network interface. The Internet address can be a host name or a standard Internet address format (e.g., 90.0.0.4). If a host name is specified, it should already have been added to the host table with `hostAdd()`.
- RETURNS** OK, or ERROR if the interface cannot be deleted.
- SEE ALSO** `ifLib`, `ifAddrGet()`, `ifDstAddrSet()`, `ifDstAddrGet()`

---

## ifAddrGet()

- NAME** `ifAddrGet()` – get the Internet address of a network interface
- SYNOPSIS**
- ```
STATUS ifAddrGet
(
    char * interfaceName,      /* name of interface, i.e. ei0 */
    char * interfaceAddress   /* buffer for Internet address */
)
```
- DESCRIPTION** This routine gets the Internet address of a specified network interface and copies it to *interfaceAddress*. This pointer should point to a buffer large enough to contain INET\_ADDR\_LEN bytes.
- RETURNS** OK or ERROR.
- SEE ALSO** `ifLib`, `ifAddrSet()`, `ifDstAddrSet()`, `ifDstAddrGet()`

---

## ifAddrSet()

**NAME** ifAddrSet() – set an interface address for a network interface

**SYNOPSIS**

```
STATUS ifAddrSet
(
    char * interfaceName,    /* name of interface to configure, i.e. ei0 */
    char * interfaceAddress /* Internet address to assign to interface */
)
```

**DESCRIPTION** This routine assigns an Internet address to a specified network interface. The Internet address can be a host name or a standard Internet address format (*e.g.*, 90.0.0.4). If a host name is specified, it should already have been added to the host table with **hostAdd()**.

A successful call to **ifAddrSet()** results in the addition of a new route.

The subnet mask used in determining the network portion of the address will be that set by **ifMaskSet()**, or the default class mask if **ifMaskSet()** has not been called. It is standard practice to call **ifMaskSet()** prior to calling **ifAddrSet()**.

**RETURNS** OK, or ERROR if the interface cannot be set.

**SEE ALSO** ifLib, ifAddrGet(), ifDstAddrSet(), ifDstAddrGet()

---

## ifAllRoutesDelete()

**NAME** ifAllRoutesDelete() – delete all routes associated with a network interface

**SYNOPSIS**

```
int ifAllRoutesDelete
(
    char * ifName,          /* name of the interface */
    int    unit             /* unit number for this interface */
)
```

**DESCRIPTION** This routine deletes all routes that have been associated with the specified interface. The routes deleted are:

- the network route added when the interface address is initialized
- the static routes added by the administrator
- ARP routes passing through the interface

Routes added by routing protocols are not deleted.

**RETURNS** The number of routes deleted, or **ERROR** if an interface is not specified.

**SEE ALSO** `ifLib`

---

## **ifBroadcastGet()**

**NAME** `ifBroadcastGet()` – get the broadcast address for a network interface

**SYNOPSIS**

```
STATUS ifBroadcastGet
(
    char * interfaceName,      /* name of interface, i.e. ei0 */
    char * broadcastAddress    /* buffer for broadcast address */
)
```

**DESCRIPTION** This routine gets the broadcast address for a specified network interface. The broadcast address is copied to the buffer *broadcastAddress*.

**RETURNS** OK or ERROR.

**SEE ALSO** `ifLib`, `ifBroadcastSet()`

---

## **ifBroadcastSet()**

**NAME** `ifBroadcastSet()` – set the broadcast address for a network interface

**SYNOPSIS**

```
STATUS ifBroadcastSet
(
    char * interfaceName,      /* name of interface to assign, i.e. ei0 */
    char * broadcastAddress    /* broadcast address to assign to interface */
)
```

**DESCRIPTION** This routine assigns a broadcast address for the specified network interface. The broadcast address must be a string in standard Internet address format (*e.g.*, 90.0.0.0).

An interface's default broadcast address is its Internet address with a host part of all ones (e.g., 90.255.255.255). This conforms to current ARPA specifications. However, some older systems use an Internet address with a host part of all zeros as the broadcast address.

---

**NOTE:** VxWorks automatically accepts a host part of all zeros as a broadcast address, in addition to the default or specified broadcast address. But if VxWorks is to broadcast to older systems using a host part of all zeros as the broadcast address, this routine should be used to change the broadcast address of the interface.

---

**RETURNS** OK or ERROR.

**SEE ALSO** ifLib

---

## ifDstAddrGet()

**NAME** ifDstAddrGet() – get the Internet address of a point-to-point peer

**SYNOPSIS**

```
STATUS ifDstAddrGet
(
    char * interfaceName,    /* name of interface, i.e. ei0 */
    char * dstAddress       /* buffer for destination address */
)
```

**DESCRIPTION** This routine gets the Internet address of a machine connected to the opposite end of a point-to-point network connection. The Internet address is copied to the buffer *dstAddress*.

**RETURNS** OK or ERROR.

**SEE ALSO** ifLib, ifDstAddrSet(), ifAddrGet()

---

## ifDstAddrSet()

- NAME** `ifDstAddrSet()` – define an address for the other end of a point-to-point link
- SYNOPSIS**
- ```
STATUS ifDstAddrSet
(
    char * interfaceName,    /* name of interface to configure, i.e. ei0 */
    char * dstAddress       /* Internet address to assign to destination */
)
```
- DESCRIPTION** This routine assigns the Internet address of a machine connected to the opposite end of a point-to-point network connection, such as a SLIP connection. Inherently, point-to-point connection-oriented protocols such as SLIP require that addresses for both ends of a connection be specified.
- RETURNS** OK or ERROR.
- SEE ALSO** `ifLib`, `ifAddrSet()`, `ifDstAddrGet()`

---

## ifFlagChange()

- NAME** `ifFlagChange()` – change the network interface flags
- SYNOPSIS**
- ```
STATUS ifFlagChange
(
    char * interfaceName,    /* name of the network interface, i.e. ei0 */
    int   flags,            /* the flag to be changed */
    BOOL  on                /* TRUE=turn on, FALSE=turn off */
)
```
- DESCRIPTION** This routine changes the flags for the specified network interfaces. If the parameter *on* is TRUE, the specified flags are turned on; otherwise, they are turned off. The routines `ifFlagGet()` and `ifFlagSet()` are called to do the actual work.
- RETURNS** OK or ERROR.
- SEE ALSO** `ifLib`, `ifAddrSet()`, `ifMaskSet()`, `ifFlagSet()`, `ifFlagGet()`

## ifFlagGet()

|                    |                                                                                                                                                                           |
|--------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>        | ifFlagGet() – get the network interface flags                                                                                                                             |
| <b>SYNOPSIS</b>    | <pre>STATUS ifFlagGet (     char * interfaceName,    /* name of the network interface, i.e. ei0 */     int * flags              /* network flags returned here */ )</pre> |
| <b>DESCRIPTION</b> | This routine gets the flags for a specified network interface. The flags are copied to the buffer <i>flags</i> .                                                          |
| <b>RETURNS</b>     | OK or ERROR.                                                                                                                                                              |
| <b>SEE ALSO</b>    | ifLib, ifFlagSet()                                                                                                                                                        |

---

## ifFlagSet()

|                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
|--------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>        | ifFlagSet() – specify the flags for a network interface                                                                                                                                                                                                                                                                                                                                                                                                      |
| <b>SYNOPSIS</b>    | <pre>STATUS ifFlagSet (     char * interfaceName,    /* name of the network interface, i.e. ei0 */     int   flags              /* network flags */ )</pre>                                                                                                                                                                                                                                                                                                  |
| <b>DESCRIPTION</b> | <p>This routine changes the flags for a specified network interface. Any combination of the following flags can be specified:</p> <p><b>IFF_UP</b> (0x1)<br/>Brings the network up or down.</p> <p><b>IFF_DEBUG</b> (0x4)<br/>Turns on debugging for the driver interface if supported.</p> <p><b>IFF_LOOPBACK</b> (0x8)<br/>Set for a loopback network.</p> <p><b>IFF_NOTRAILERS</b> (0x20)<br/>Always set (VxWorks does not use the trailer protocol).</p> |

**IFF\_PROMISC** (0x100)

Tells the driver to accept all packets, not just broadcast packets and packets addressed to itself.

**IFF\_ALLMULTI** (0x200)

Tells the driver to accept all multicast packets.

**IFF\_NOARP** (0x80)

Disables ARP for the interface.

---

**NOTE:** The following flags can only be set at interface initialization time. Specifying these flags does not change any settings in the interface data structure.

---

**IFF\_POINTOPOINT** (0x10)

Identifies a point-to-point interface such as PPP or SLIP.

**IFF\_RUNNING** (0x40)

Set when the device turns on.

**IFF\_BROADCAST** (0x2)

Identifies a broadcast interface.

**RETURNS** OK or ERROR.

**SEE ALSO** `ifLib`, `ifFlagChange()`, `ifFlagGet()`

---

## ifIndexAlloc()

**NAME** `ifIndexAlloc()` – return a unique interface index

**SYNOPSIS** `int ifIndexAlloc (void)`

**DESCRIPTION** `ifIndexAlloc()` returns a unique integer to be used as an interface index. The first index returned is 1. **ERROR** is returned if the library has not been initialized by a call to `ifIndexLibInit()`.

**RETURNS** interface index or **ERROR**

**SEE ALSO** `ifIndexLib`

## **ifIndexLibInit()**

|                    |                                                                                                                                 |
|--------------------|---------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>        | <b>ifIndexLibInit()</b> – initializes library variables                                                                         |
| <b>SYNOPSIS</b>    | <code>void ifIndexLibInit (void)</code>                                                                                         |
| <b>DESCRIPTION</b> | <b>ifIndexLibInit()</b> resets library internal state. This function must be called before any other functions in this library. |
| <b>RETURNS</b>     | N/A                                                                                                                             |
| <b>SEE ALSO</b>    | <b>ifIndexLib</b>                                                                                                               |

---

## **ifIndexLibShutdown()**

|                    |                                                                                                                                             |
|--------------------|---------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>        | <b>ifIndexLibShutdown()</b> – frees library variables                                                                                       |
| <b>SYNOPSIS</b>    | <code>void ifIndexLibShutdown (void)</code>                                                                                                 |
| <b>DESCRIPTION</b> | <b>ifIndexLibShutdown()</b> frees library internal structures. <b>ifIndexLibInit()</b> must be called before the library can be used again. |
| <b>RETURNS</b>     | N/A                                                                                                                                         |
| <b>SEE ALSO</b>    | <b>ifIndexLib</b>                                                                                                                           |

---

## ifIndexTest()

|                    |                                                                                                                                                                                                                                                                                                                                            |
|--------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>        | <code>ifIndexTest()</code> – returns true if an index has been allocated.                                                                                                                                                                                                                                                                  |
| <b>SYNOPSIS</b>    | <pre>BOOL ifIndexTest (     int ifIndex          /* the index to test */ )</pre>                                                                                                                                                                                                                                                           |
| <b>DESCRIPTION</b> | <code>ifIndexTest()</code> returns <code>TRUE</code> if <i>index</i> has already been allocated by <code>ifIndexLibAlloc()</code> . Otherwise returns <code>FALSE</code> . If the library has not been initialized returns <code>FALSE</code> . This function does not check if the index actually belongs to a currently valid interface. |
| <b>RETURNS</b>     | <code>TRUE</code> or <code>FALSE</code>                                                                                                                                                                                                                                                                                                    |
| <b>SEE ALSO</b>    | <code>ifIndexLib</code>                                                                                                                                                                                                                                                                                                                    |

---

## ifIndexToIfName()

|                    |                                                                                                                                                                                                                             |
|--------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>        | <code>ifIndexToIfName()</code> – returns the interface name given the interface index                                                                                                                                       |
| <b>SYNOPSIS</b>    | <pre>STATUS ifIndexToIfName (     unsigned short ifIndex, /* Interface index */     char *         ifName   /* Where the name is to be stored */ )</pre>                                                                    |
| <b>DESCRIPTION</b> | This routine returns the interface name for the interface referenced by the <i>ifIndex</i> parameter.<br><i>ifIndex</i><br>The index for the interface.<br><i>ifName</i><br>The location where the interface name is copied |
| <b>RETURNS</b>     | <code>OK</code> on success, <code>ERROR</code> otherwise.                                                                                                                                                                   |
| <b>SEE ALSO</b>    | <code>ifLib</code>                                                                                                                                                                                                          |

**ifMaskGet()**

---

## ifMaskGet()

**NAME** ifMaskGet() – get the subnet mask for a network interface

**SYNOPSIS**

```
STATUS ifMaskGet
(
    char * interfaceName,    /* name of interface, i.e. ei0 */
    int * netMask           /* buffer for subnet mask */
)
```

**DESCRIPTION** This routine gets the subnet mask for a specified network interface. The subnet mask is copied to the buffer *netMask*. The subnet mask is returned in host byte order.

**RETURNS** OK or ERROR.

**SEE ALSO** ifLib, ifAddrGet(), ifFlagGet()

---

## ifMaskSet()

**NAME** ifMaskSet() – define a subnet for a network interface

**SYNOPSIS**

```
STATUS ifMaskSet
(
    char * interfaceName,    /* name of interface to set mask for, i.e. ei0 */
    int netMask             /* subnet mask (e.g. 0xff000000) */
)
```

**DESCRIPTION** This routine allocates additional bits to the network portion of an Internet address. The network portion is specified with a mask that must contain ones in all positions that are to be interpreted as the network portion. This includes all the bits that are normally interpreted as the network portion for the given class of address, plus the bits to be added. Note that all bits must be contiguous. The mask is specified in host byte order.

In order to correctly interpret the address, a subnet mask should be set for an interface prior to setting the Internet address of the interface with the routine **ifAddrSet()**.

**RETURNS** OK or ERROR.

**SEE ALSO** ifLib, ifAddrSet()

---

## ifMetricGet()

**NAME** `ifMetricGet()` – get the metric for a network interface

**SYNOPSIS**

```
STATUS ifMetricGet
(
    char * interfaceName,    /* name of the network interface, i.e. ei0 */
    int * pMetric           /* returned interface's metric */
)
```

**DESCRIPTION** This routine retrieves the metric for a specified network interface. The metric is copied to the buffer *pMetric*.

**RETURNS** OK or ERROR.

**SEE ALSO** `ifLib`, `ifMetricSet()`

---

## ifMetricSet()

**NAME** `ifMetricSet()` – specify a network interface hop count

**SYNOPSIS**

```
STATUS ifMetricSet
(
    char * interfaceName,    /* name of the network interface, i.e. ei0 */
    int   metric            /* metric for this interface */
)
```

**DESCRIPTION** This routine configures *metric* for a network interface from the host machine to the destination network. This information is used primarily by the IP routing algorithm to compute the relative distance for a collection of hosts connected to each interface. For example, a higher *metric* for SLIP interfaces can be specified to discourage routing a packet to slower serial line connections. Note that when *metric* is zero, the IP routing algorithm allows for the direct sending of a packet having an IP network address that is not necessarily the same as the local network address.

**RETURNS** OK or ERROR.

**SEE ALSO** `ifLib`, `ifMetricGet()`

## ifNameToIfIndex()

|                    |                                                                                                                                                                                   |
|--------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>        | <b>ifNameToIfIndex()</b> – returns the interface index given the interface name                                                                                                   |
| <b>SYNOPSIS</b>    | <pre>unsigned short ifNameToIfIndex (     char * ifName          /* a string describing the full interface */                           /* name. e.g., "fei0" */ )</pre>          |
| <b>DESCRIPTION</b> | This routine returns the interface index for the interface named by the <i>ifName</i> parameter, which provides a string describing the full interface name. For example, “fei0”. |
| <b>RETURNS</b>     | The interface index, if the interface could be located, 0, otherwise. 0 is not a valid value for interface index.                                                                 |
| <b>SEE ALSO</b>    | <b>ifLib</b>                                                                                                                                                                      |

---

## ifRouteDelete()

|                    |                                                                                                                                                                                                                                                                                                       |
|--------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>        | <b>ifRouteDelete()</b> – delete routes associated with a network interface                                                                                                                                                                                                                            |
| <b>SYNOPSIS</b>    | <pre>int ifRouteDelete (     char * ifName,        /* name of the interface */     int   unit            /* unit number for this interface */ )</pre>                                                                                                                                                 |
| <b>DESCRIPTION</b> | This routine deletes all routes that have been associated with the specified interface. A route is associated with an interface if its destination equals to the assigned address, or network number. This routine does not remove routes to arbitrary destinations that through the given interface. |
| <b>RETURNS</b>     | The number of routes deleted, or <b>ERROR</b> if an interface is not specified.                                                                                                                                                                                                                       |
| <b>SEE ALSO</b>    | <b>ifLib</b>                                                                                                                                                                                                                                                                                          |

---

## ifShow()

**NAME** `ifShow()` – display the attached network interfaces

**SYNOPSIS**

```
void ifShow
(
    char * ifName          /* name of the interface to show */
)
```

**DESCRIPTION**

This routine displays the attached network interfaces for debugging and diagnostic purposes. If *ifName* is given, only the interfaces belonging to that group are displayed. If *ifName* is omitted, all attached interfaces are displayed.

For each interface selected, the following are shown: Internet address, point-to-point peer address (if using SLIP), broadcast address, netmask, subnet mask, Ethernet address, route metric, maximum transfer unit, number of packets sent and received on this interface, number of input and output errors, and flags (such as loopback, point-to-point, broadcast, promiscuous, ARP, running, and debug).

**EXAMPLE**

The following call displays all interfaces whose names begin with “ln”, (such as “ln0”, “ln1”, and “ln2”):

```
-> ifShow "ln"
```

The following call displays just the interface “ln0”:

```
-> ifShow "ln0"
```

**RETURNS** N/A

**SEE ALSO** `netShow`, `routeShow()`, `ifLib`

---

## ifunit()

**NAME** `ifunit()` – map an interface name to an interface structure pointer

**SYNOPSIS**

```
struct ifnet *ifunit
(
    char * ifname          /* name of the interface */
)
```

**DESCRIPTION** This routine returns a pointer to a network interface structure for *name* or **NULL** if no such interface exists. For example:

```
    struct ifnet *pIf;  
    ...  
    pIf = ifunit ("ln0");
```

*pIf* points to the data structure that describes the first network interface device if *ln0* is mapped successfully.

**RETURNS** A pointer to the interface structure, or **NULL** if an interface is not found.

**SEE ALSO** [ifLib](#)

---

## ifUnnumberedSet()

**NAME** `ifUnnumberedSet()` – configure an interface to be unnumbered

**SYNOPSIS** `STATUS ifUnnumberedSet`

```
(  
    char * pIfName,          /* Name of interface to configure */  
    char * pDstIp,          /* Destination address of the point to */  
                                /* point link */  
    char * pBorrowedIp,     /* The borrowed IP address/router ID */  
    char * pDstMac          /* Destination MAC address */  
)
```

**DESCRIPTION** This API sets an interface unnumbered. It sets the `IFF_POINTOPOINT` flags and creates a routing entry through the interface using a user-specified destination IP address. The unnumbered link can then be uniquely referred to by the destination IP address, *pDstIp*, when adding routes. The interface is assigned a “borrowed” IP address—borrowed from another interface on the machine. In RFC 1812 it is also called the router ID. This address will be used to generate any needed ICMP messages or the like. Note that ARP is not able to run on an unnumbered link.

The initialization of the unnumbered device is similar to other network devices, but it does have a few additional steps and concerns. `ifUnnumberedSet()` must come next after `ipAttach()`. Please note that the interface using the IP address that the unnumbered interface will borrow must be brought up first and configured with `ifAddrSet` or equivalent. This is required to ensure normal network operation for that IP address/interface. After `ifUnnumberedSet()`, one must create additional routing entries (using `mRouteAdd()`, `routeNetAdd()`, *etc.*) in order to reach other networks, including the network to which the destination IP address belongs.

The *pDstMac* field in **ifUnnumberedSet()** is used to specify the destination's MAC address. It should be left **NULL** if the destination is not an Ethernet device. If the MAC address is not known, then supply an artificial address. We recommend using "00:00:00:00:00:01". The destination interface can then be set promiscuous to accept this artificial address. This is accomplished using the *ifpromisc* command.

Example:

```
ipAttach (1, "fei")
ifUnnumberedSet ("fei1", "120.12.12.12", "140.34.78.94", "00:a0:d0:d8:c8:14")
routeNetAdd ("120.12.0.0","120.12.12.12") <One possible network>
routeNetAdd ("178.45.0.0","120.12.12.12") <Another possible network>
```

**RETURNS** OK, or **ERROR** if the interface cannot be set.

**SEE ALSO** ifLib

---

## igmpShowInit()

**NAME** igmpShowInit() – initialize IGMP show routines

**SYNOPSIS** void igmpShowInit (void)

**DESCRIPTION** This routine links the IGMP show facility into the VxWorks system. These routines are included automatically if **INCLUDE\_NET\_SHOW** and **INCLUDE\_IGMP** are defined.

**RETURNS** N/A

**SEE ALSO** igmpShow

## **igmpstatShow()**

|                    |                                                                                              |
|--------------------|----------------------------------------------------------------------------------------------|
| <b>NAME</b>        | <b>igmpstatShow()</b> – display statistics for IGMP                                          |
| <b>SYNOPSIS</b>    | <code>void igmpstatShow (void)</code>                                                        |
| <b>DESCRIPTION</b> | This routine displays statistics for the IGMP (Internet Group Management Protocol) protocol. |
| <b>RETURNS</b>     | N/A                                                                                          |
| <b>SEE ALSO</b>    | <b>igmpShow</b>                                                                              |

---

## **index()**

|                    |                                                                                                                                                               |
|--------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>        | <b>index()</b> – find the first occurrence of a character in a string                                                                                         |
| <b>SYNOPSIS</b>    | <pre>char *index (     const char * s,          /* string in which to find character */     int      c              /* character to find in string */ )</pre> |
| <b>DESCRIPTION</b> | This routine finds the first occurrence of character <i>c</i> in string <i>s</i> .                                                                            |
| <b>RETURNS</b>     | A pointer to the located character, or NULL if <i>c</i> is not found.                                                                                         |
| <b>SEE ALSO</b>    | <b>bLib</b> , <b>strchr()</b>                                                                                                                                 |

---

## inet\_addr()

**NAME** `inet_addr()` – convert a dot notation Internet address to a long integer

**SYNOPSIS**

```
u_long inet_addr
(
    char * inetString      /* string inet address */
)
```

**DESCRIPTION** This routine interprets an Internet address. All the network library routines call this routine to interpret entries in the data bases which are expected to be an address. The value returned is in network order. Numbers will be interpreted as octal if preceded by a zero (e.g., “017.0.0.3”), as hexadecimal if preceded by 0x (e.g., “0x17.0.0.4”), and as decimal in all other cases.

**EXAMPLE** The following example returns 0x5a000002:

```
inet_addr ("90.0.0.2");
```

**RETURNS** The Internet address, or **ERROR**.

**SEE ALSO** `inetLib`

---

## inet\_aton()

**NAME** `inet_aton()` – convert a network address from dot notation, store in a structure

**SYNOPSIS**

```
STATUS inet_aton
(
    char *          pString,    /* string containing address, dot notation */
    struct in_addr * inetAddress /* struct in which to store address */
)
```

**DESCRIPTION** This routine interprets an Internet address. All the network library routines call this routine to interpret entries in the data bases that are expected to be an address. The value returned is stored in network byte order in the structure provided.

**EXAMPLE** The following example returns 0x5a000002 in the `s_addr` member of the structure pointed to by `pinetAddr`:

```
inet_aton ("90.0.0.2", pinetAddr);
```

**RETURNS** OK, or ERROR if address is invalid.

**SEE ALSO** inetLib

---

## inet\_lnaof()

**NAME** *inet\_lnaof()* – get the local address (host number) from the Internet address

**SYNOPSIS**

```
int inet_lnaof
(
    int inetAddress          /* inet addr from which to extract local */
                              /* portion */
)
```

**DESCRIPTION** This routine returns the local network address portion of an Internet address. The routine handles class A, B, and C network number formats.

**EXAMPLE** The following example returns 2:

```
inet_lnaof (0x5a000002);
```

**RETURNS** The local address portion of *inetAddress*.

**SEE ALSO** inetLib

---

## inet\_makeaddr()

**NAME** *inet\_makeaddr()* – form an Internet address from network and host numbers

**SYNOPSIS**

```
struct in_addr inet_makeaddr
(
    int netAddr,             /* network part of the address */
    int hostAddr            /* host part of the address */
)
```

**DESCRIPTION** This routine constructs the Internet address from the network number and local host address.

---

**WARNING:** This routine is supplied for UNIX compatibility only. Each time this routine is called, four bytes are allocated from memory. Use `inet_makeaddr_b()` instead.

---

**EXAMPLE** The following example returns the address 0x5a000002 to the structure `in_addr`:

```
inet_makeaddr (0x5a, 2);
```

**RETURNS** The network address in an `in_addr` structure.

**SEE ALSO** `inetLib`, `inet_makeaddr_b()`

---

## `inet_makeaddr_b()`

**NAME** `inet_makeaddr_b()` – form an Internet address from network and host numbers

**SYNOPSIS**

```
void inet_makeaddr_b  
(  
    int          netAddr, /* network part of the inet address */  
    int          hostAddr, /* host part of the inet address */  
    struct in_addr * pInetAddr /* where to return the inet address */  
)
```

**DESCRIPTION** This routine constructs the Internet address from the network number and local host address. This routine is identical to the UNIX `inet_makeaddr()` routine except that you must provide a buffer for the resulting value.

**EXAMPLE** The following copies the address 0x5a000002 to the location pointed to by `pInetAddr`:

```
inet_makeaddr_b (0x5a, 2, pInetAddr);
```

**RETURNS** N/A

**SEE ALSO** `inetLib`

---

## inet\_netof()

|                    |                                                                                 |
|--------------------|---------------------------------------------------------------------------------|
| <b>NAME</b>        | <code>inet_netof()</code> – return the network number from an Internet address  |
| <b>SYNOPSIS</b>    | <pre>int inet_netof (     struct in_addr inetAddress /* inet address */ )</pre> |
| <b>DESCRIPTION</b> | This routine extracts the network portion of an Internet address.               |
| <b>EXAMPLE</b>     | The following example returns 0x5a:<br><pre>inet_netof (0x5a000002);</pre>      |
| <b>RETURNS</b>     | The network portion of <i>inetAddress</i> .                                     |
| <b>SEE ALSO</b>    | <code>inetLib</code>                                                            |

---

## inet\_netof\_string()

|                    |                                                                                                                                                                                                                                                                                                                                                                                 |
|--------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>        | <code>inet_netof_string()</code> – extract the network address in dot notation                                                                                                                                                                                                                                                                                                  |
| <b>SYNOPSIS</b>    | <pre>void inet_netof_string (     char * inetString,          /* inet addr to extract local portion from */     char * netString           /* net inet address to return */ )</pre>                                                                                                                                                                                             |
| <b>DESCRIPTION</b> | This routine extracts the network Internet address from a host Internet address (specified in dotted decimal notation). The routine handles class A, B, and C network addresses. The buffer <i>netString</i> should be <code>INET_ADDR_LEN</code> bytes long.<br><hr/> <b>NOTE:</b> This is the only routine in <code>inetLib</code> that handles subnet masks correctly. <hr/> |
| <b>EXAMPLE</b>     | The following example copies “90.0.0.0” to <i>netString</i> :<br><pre>inet_netof_string ("90.0.0.2", netString);</pre>                                                                                                                                                                                                                                                          |
| <b>RETURNS</b>     | N/A                                                                                                                                                                                                                                                                                                                                                                             |
| <b>SEE ALSO</b>    | <code>inetLib</code>                                                                                                                                                                                                                                                                                                                                                            |

---

## inet\_network()

|                    |                                                                                                  |
|--------------------|--------------------------------------------------------------------------------------------------|
| <b>NAME</b>        | <code>inet_network()</code> – convert an Internet network number from string to address          |
| <b>SYNOPSIS</b>    | <pre>u_long inet_network (     char * inetString      /* string version of inet addr */ )</pre>  |
| <b>DESCRIPTION</b> | This routine forms a network address from an ASCII string containing an Internet network number. |
| <b>EXAMPLE</b>     | The following example returns 0x5a:<br><pre>inet_network ("90");</pre>                           |
| <b>RETURNS</b>     | The Internet address for an ASCII string, or <b>ERROR</b> if invalid.                            |
| <b>SEE ALSO</b>    | <code>inetLib</code>                                                                             |

---

## inet\_ntoa()

|                    |                                                                                                                                                                                                                                                                                |
|--------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>        | <code>inet_ntoa()</code> – convert a network address to dotted decimal notation                                                                                                                                                                                                |
| <b>SYNOPSIS</b>    | <pre>char *inet_ntoa (     struct in_addr inetAddress /* inet address */ )</pre>                                                                                                                                                                                               |
| <b>DESCRIPTION</b> | This routine converts an Internet address in network format to dotted decimal notation.<br><b>WARNING:</b> This routine is supplied for UNIX compatibility only. Each time this routine is called, 18 bytes are allocated from memory. Use <code>inet_ntoa_b()</code> instead. |
| <b>EXAMPLE</b>     | The following example returns a pointer to the string “90.0.0.2”:<br><pre>struct in_addr iaddr; ... iaddr.s_addr = 0x5a000002; ... inet_ntoa (iaddr);</pre>                                                                                                                    |

**RETURNS** A pointer to the string version of an Internet address.

**SEE ALSO** `inetLib`, `inet_ntoa_b()`

---

## **inet\_ntoa\_b()**

**NAME** `inet_ntoa_b()` – convert an network address to dot notation, store it in a buffer

**SYNOPSIS**

```
void inet_ntoa_b
(
    struct in_addr inetAddress, /* inet address */
    char *          pString     /* where to return ASCII string */
)
```

**DESCRIPTION** This routine converts an Internet address in network format to dotted decimal notation. This routine is identical to the UNIX `inet_ntoa()` routine except that you must provide a buffer of size `INET_ADDR_LEN`.

**EXAMPLE** The following example copies the string “90.0.0.2” to `pString`:

```
struct in_addr iaddr;
...
iaddr.s_addr = 0x5a000002;
...
inet_ntoa_b (iaddr, pString);
```

**RETURNS** N/A

**SEE ALSO** `inetLib`

---

## inetstatShow()

|                    |                                                                                                                                                                                                                                                                                 |
|--------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>        | <code>inetstatShow()</code> – display all active connections for Internet protocol sockets                                                                                                                                                                                      |
| <b>SYNOPSIS</b>    | <code>void inetstatShow (void)</code>                                                                                                                                                                                                                                           |
| <b>DESCRIPTION</b> | <p>This routine displays a list of all active Internet protocol sockets in a format similar to the UNIX <code>netstat</code> command.</p> <p>If you want <code>inetstatShow()</code> to display TCP socket status, then <code>INCLUDE_TCP_SHOW</code> needs to be included.</p> |
| <b>RETURNS</b>     | N/A                                                                                                                                                                                                                                                                             |
| <b>SEE ALSO</b>    | <code>netShow</code>                                                                                                                                                                                                                                                            |

---

## infinity()

|                      |                                                           |
|----------------------|-----------------------------------------------------------|
| <b>NAME</b>          | <code>infinity()</code> – return a very large double      |
| <b>SYNOPSIS</b>      | <code>double infinity (void)</code>                       |
| <b>DESCRIPTION</b>   | This routine returns a very large double.                 |
| <b>INCLUDE FILES</b> | <code>math.h</code>                                       |
| <b>RETURNS</b>       | The double-precision representation of positive infinity. |
| <b>SEE ALSO</b>      | <code>mathALib</code>                                     |

## **infinityf()**

|                      |                                                           |
|----------------------|-----------------------------------------------------------|
| <b>NAME</b>          | <b>infinityf()</b> – return a very large float            |
| <b>SYNOPSIS</b>      | <code>float infinityf (void)</code>                       |
| <b>DESCRIPTION</b>   | This routine returns a very large float.                  |
| <b>INCLUDE FILES</b> | <code>math.h</code>                                       |
| <b>RETURNS</b>       | The single-precision representation of positive infinity. |
| <b>SEE ALSO</b>      | <code>mathALib</code>                                     |

---

## **inflate()**

|                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
|--------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>        | <b>inflate()</b> – inflate compressed code                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
| <b>SYNOPSIS</b>    | <pre>int inflate (     Byte * src,     Byte * dest,     int   nBytes )</pre>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| <b>DESCRIPTION</b> | This routine inflates <i>nBytes</i> of data starting at address <i>src</i> . The inflated code is copied starting at address <i>dest</i> . Two sanity checks are performed on the data being decompressed. First, we look for a magic number at the start of the data to verify that it is really a compressed stream. Second, the entire data is optionally check-summed to verify its integrity. By default, the checksum is not verified in order to speed up the booting process. To turn on checksum verification, set the global variable <b>inflateCksum</b> to <b>TRUE</b> in the BSP. |
| <b>RETURNS</b>     | OK or ERROR.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| <b>SEE ALSO</b>    | <code>inflateLib</code>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |

---

## intConnect()

|                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
|--------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>        | <b>intConnect()</b> – connect a C routine to a hardware interrupt                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
| <b>SYNOPSIS</b>    | <pre>STATUS intConnect (     VOIDFUNCPTR * vector,      /* interrupt vector to attach to */     VOIDFUNCPTR  routine,     /* routine to be called */     int          parameter    /* parameter to be passed to routine */ )</pre>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| <b>DESCRIPTION</b> | <p>This routine connects a specified C routine to a specified interrupt vector. The address of <i>routine</i> is generally stored at <i>vector</i> so that <i>routine</i> is called with <i>parameter</i> when the interrupt occurs. The routine is invoked in supervisor mode at interrupt level. A proper C environment is established, the necessary registers saved, and the stack set up.</p> <p>The routine can be any normal C code, except that it must not invoke certain operating system functions that may block or perform I/O operations.</p> <p>This routine generally simply calls <b>intHandlerCreate()</b> and <b>intVecSet()</b>. The address of the handler returned by <b>intHandlerCreate()</b> is what actually goes in the interrupt vector.</p> <p>This routine takes an interrupt vector as a parameter, which is the byte offset into the vector table. Macros are provided to convert between interrupt vectors and interrupt numbers, see <b>intArchLib</b>.</p> |
| <b>NOTE ARM</b>    | ARM processors generally do not have on-chip interrupt controllers. Control of interrupts is a BSP-specific matter. This routine calls a BSP-specific routine to install the handler such that, when the interrupt occurs, <i>routine</i> is called with <i>parameter</i> .                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| <b>NOTE X86</b>    | Refer to the special x86 routine <b>intHandlerCreateI86()</b> .                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
| <b>NOTE SH</b>     | <p>The on-chip interrupt controller (INTC) design of SH architecture depends on the processor type, but there are some similarities. The number of external interrupt inputs are limited, so it may necessary to multiplex some interrupt requests. However most of them are auto-vectored, thus have only one vector to an external interrupt input. As a framework to handle this type of multiplexed interrupt, you can use your original <b>intConnect()</b> code by hooking it to <b>_func_intConnectHook</b> pointer. If <b>_func_intConnectHook</b> is set, the SH version of <b>intConnect()</b> simply calls the hooked routine with same arguments, then returns the status of hooked routine. A <b>sysLib</b> sample is shown below:</p> <pre>#include "intLib.h" #include "iv.h"          /* INUM_INTR_HIGH for SH7750/SH7700 */ #define SYS_INT_TBL_SIZE (255 - INUM_INTR_HIGH) typedef struct</pre>                                                                             |

```
{
VOIDFUNCPTR routine;      /* routine to be called */
int      parameter;      /* parameter to be passed */
} SYS_INT_TBL;
LOCAL SYS_INT_TBL sysIntTbl [SYS_INT_TBL_SIZE]; /* local vector table */
LOCAL int sysInumVirtBase = INUM_INTR_HIGH + 1;
STATUS sysIntConnect
(
VOIDFUNCPTR *vec,          /* interrupt vector to attach to */
VOIDFUNCPTR routine,      /* routine to be called */
int      param            /* parameter to be passed to routine */
)
{
FUNCPTR intDrvRtn;

if (vec >= INUM_TO_IVEC (0) && vec < INUM_TO_IVEC (sysInumVirtBase))
{
/* do regular intConnect() process */
intDrvRtn = intHandlerCreate ((FUNCPTR) routine, param);

if (intDrvRtn == NULL)
return ERROR;

/* make vector point to synthesized code */

intVecSet ((FUNCPTR *) vec, (FUNCPTR) intDrvRtn);
}
else
{
int index = IVEC_TO_INUM (vec) - sysInumVirtBase;

if (index < 0 || index >= SYS_INT_TBL_SIZE)
return ERROR;

sysIntTbl [index].routine = routine;
sysIntTbl [index].parameter = param;
}

return OK;
}
void sysHwInit (void)
{
...
_func_intConnectHook = (FUNCPTR) sysIntConnect;
}
LOCAL void sysVmeIntr (void)
```

```

{
volatile UINT32 vec = *VME_VEC_REGISTER; /* get VME interrupt vector */
int i = vec - sysInumVirtBase;

if (i >= 0 && i < SYS_INT_TBL_SIZE && sysIntTbl[i].routine != NULL)
    (*sysIntTbl[i].routine)(sysIntTbl[i].parameter);
else
    logMsg ("uninitialized VME interrupt: vec = %d\n", vec,0,0,0,0);
}
void sysHwInit2 (void)
{
int i;
...
/* initialize VME interrupts dispatch table */
for (i = 0; i < SYS_INT_TBL_SIZE; i++)
    {
        sysIntTbl[i].routine = (VOIDFUNCPTR) NULL;
        sysIntTbl[i].parameter = NULL;
    }
/* connect generic VME interrupts handler */
intConnect (INT_VEC_VME, sysVmeIntr, NULL);
...
}

```

The used vector numbers of SH processors are limited to certain ranges, depending on the processor type. The `sysInumVirtBase` should be initialized to a value higher than the last used vector number, defined as `INUM_INTR_HIGH`. It is typically safe to set `sysInumVirtBase` to `(INUM_INTR_HIGH + 1)`.

The `sysIntConnect()` routine simply acts as the regular `intConnect()` if *vector* is smaller than `INUM_TO_IVEC (sysInumVirtBase)`, so `sysHwInit2()` connects a common VME interrupt dispatcher `sysVmeIntr` to the multiplexed interrupt vector. If *vector* is equal to or greater than `INUM_TO_IVEC (sysInumVirtBase)`, the `sysIntConnect()` fills a local vector entry in `sysIntTbl[]` with an individual VME interrupt handler, in a coordinated manner with `sysVmeIntr`.

**RETURNS** OK, or ERROR if the interrupt handler cannot be built.

**SEE ALSO** `intArchLib`, `intHandlerCreate()`, `intVecSet()`

## intContext()

|                    |                                                                                                                                    |
|--------------------|------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>        | <b>intContext()</b> – determine if the current state is in interrupt or task context                                               |
| <b>SYNOPSIS</b>    | <b>BOOL intContext (void)</b>                                                                                                      |
| <b>DESCRIPTION</b> | This routine returns <b>TRUE</b> only if the current execution state is in interrupt context and not in a meaningful task context. |
| <b>RETURNS</b>     | <b>TRUE</b> or <b>FALSE</b> .                                                                                                      |
| <b>SEE ALSO</b>    | <b>intLib</b>                                                                                                                      |

---

## intCount()

|                    |                                                                          |
|--------------------|--------------------------------------------------------------------------|
| <b>NAME</b>        | <b>intCount()</b> – get the current interrupt nesting depth              |
| <b>SYNOPSIS</b>    | <b>int intCount (void)</b>                                               |
| <b>DESCRIPTION</b> | This routine returns the number of interrupts that are currently nested. |
| <b>RETURNS</b>     | The number of nested interrupts.                                         |
| <b>SEE ALSO</b>    | <b>intLib</b>                                                            |

---

## intCRGet()

|                    |                                                                         |
|--------------------|-------------------------------------------------------------------------|
| <b>NAME</b>        | <b>intCRGet()</b> – read the contents of the cause register (MIPS)      |
| <b>SYNOPSIS</b>    | <b>int intCRGet (void)</b>                                              |
| <b>DESCRIPTION</b> | This routine reads and returns the contents of the MIPS cause register. |
| <b>RETURNS</b>     | The contents of the cause register.                                     |
| <b>SEE ALSO</b>    | <b>intArchLib</b>                                                       |

---

## intCRSet()

|                    |                                                                                            |
|--------------------|--------------------------------------------------------------------------------------------|
| <b>NAME</b>        | <code>intCRSet()</code> – write the contents of the cause register (MIPS)                  |
| <b>SYNOPSIS</b>    | <pre>void intCRSet (     int value          /* value to write to cause register */ )</pre> |
| <b>DESCRIPTION</b> | This routine writes the contents of the MIPS cause register.                               |
| <b>RETURNS</b>     | N/A                                                                                        |
| <b>SEE ALSO</b>    | <code>intArchLib</code>                                                                    |

---

## intDisable()

|                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
|--------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>        | <code>intDisable()</code> – disable corresponding interrupt bits (MIPS, PowerPC, ARM)                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
| <b>SYNOPSIS</b>    | <pre>int intDisable (     int level          /* new interrupt bits (0x0 - 0xff00) */ )</pre>                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| <b>DESCRIPTION</b> | <p>On MIPS and PowerPC architectures, this routine disables the corresponding interrupt bits from the present status register.</p> <hr/> <p><b>NOTE:</b> ARM processors generally do not have on-chip interrupt controllers. Control of interrupts is a BSP-specific matter. This routine calls a BSP-specific routine to disable a particular interrupt level, regardless of the current interrupt mask level.</p> <hr/> <p><b>NOTE:</b> For MIPS, the macros <code>SR_IBIT1 - SR_IBIT8</code> define bits that may be set.</p> <hr/> |
| <b>RETURNS</b>     | OK or ERROR. (MIPS: The previous contents of the status register).                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
| <b>SEE ALSO</b>    | <code>intArchLib</code>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |

---

## intEnable()

|                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
|--------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>        | <b>intEnable()</b> – enable corresponding interrupt bits (MIPS, PowerPC, ARM)                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
| <b>SYNOPSIS</b>    | <pre>int intEnable (     int level                /* new interrupt bits (0x00 - 0xff00) */ )</pre>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
| <b>DESCRIPTION</b> | <p>This routine enables the input interrupt bits on the present status register of the MIPS and PowerPC processors.</p> <hr/> <p><b>NOTE:</b> ARM processors generally do not have on-chip interrupt controllers. Control of interrupts is a BSP-specific matter. This routine calls a BSP-specific routine to enable the interrupt. For each interrupt level to be used, there must be a call to this routine before it will be allowed to interrupt.</p> <hr/> <p><b>NOTE:</b> For MIPS, it is strongly advised that the level be a combination of <b>SR_IBIT1</b> - <b>SR_IBIT8</b>.</p> <hr/> |
| <b>RETURNS</b>     | OK or ERROR. (MIPS: The previous contents of the status register).                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
| <b>SEE ALSO</b>    | <b>intArchLib</b>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |

---

## intHandlerCreate()

|                    |                                                                                                                                                                                                                                                                                                                                                                  |
|--------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>        | <b>intHandlerCreate()</b> – construct an interrupt handler for a C routine (68K, x86, MIPS, SimSolaris)                                                                                                                                                                                                                                                          |
| <b>SYNOPSIS</b>    | <pre>FUNCPTR intHandlerCreate (     FUNCPTR routine,        /* routine to be called */     int    parameter       /* parameter to be passed to routine */ )</pre>                                                                                                                                                                                                |
| <b>DESCRIPTION</b> | <p>This routine builds an interrupt handler around the specified C routine. This interrupt handler is then suitable for connecting to a specific vector address with <b>intVecSet()</b>. The interrupt handler is invoked in supervisor mode at interrupt level. A proper C environment is established, the necessary registers saved, and the stack set up.</p> |

The routine can be any normal C code, except that it must not invoke certain operating system functions that may block or perform I/O operations.

**RETURNS** A pointer to the new interrupt handler, or NULL if memory is insufficient.

**SEE ALSO** `intArchLib`

---

## intHandlerCreateI86()

**NAME** `intHandlerCreateI86()` – construct an interrupt handler for a C routine (x86)

**SYNOPSIS**

```

FUNCPTR intHandlerCreateI86
(
    FUNCPTR routine,           /* routine to be called */
    int     parameter,        /* parameter to be passed to routine */
    FUNCPTR routineBoi,       /* BOI routine to be called */
    int     parameterBoi,     /* parameter to be passed to routineBoi */
    FUNCPTR routineEoi,       /* EOI routine to be called */
    int     parameterEoi      /* parameter to be passed to routineEoi */
)

```

**DESCRIPTION** This routine builds an interrupt handler around a specified C routine. This interrupt handler is then suitable for connecting to a specific vector address with `intVecSet()`. The interrupt handler is invoked in supervisor mode at interrupt level. A proper C environment is established, the necessary registers saved, and the stack set up.

The routine can be any normal C code, except that it must not invoke certain operating system functions that may block or perform I/O operations.

**IMPLEMENTATION** This routine builds an interrupt handler of the following form in allocated memory:

```

00 e8 kk kk kk kk      call    _intEnt      * tell kernel
05 50                  pushl   %eax         * save regs
06 52                  pushl   %edx
07 51                  pushl   %ecx
08 68 pp pp pp pp     pushl   $_parameterBoi * push BOI param
13 e8 rr rr rr rr     call    _routineBoi  * call BOI routine
18 68 pp pp pp pp     pushl   $_parameter  * push param
23 e8 rr rr rr rr     call    _routine     * call C routine
28 68 pp pp pp pp     pushl   $_parameterEoi * push EOI param
33 e8 rr rr rr rr     call    _routineEoi  * call EOI routine
38 83 c4 0c           addl   $12, %esp     * pop param
41 59                  popl    %ecx         * restore regs

```

```
42 5a                popl    %edx
43 58                popl    %eax
44 e9 kk kk kk kk   jmp     _intExit      * exit via kernel
```

Third and fourth parameter of **intHandlerCreateI86()** are the BOI routine address and its parameter that are inserted into the code as “routineBoi” and “parameterBoi”. Fifth and sixth parameter of **intHandlerCreateI86()** are the EOI routine address and its parameter that are inserted into the code as “routineEoi” and “parameterEoi”. The BOI routine detects if this interrupt is stray/spurious/phantom by interrogating the interrupt controller, and returns from the interrupt if it is. The EOI routine issues End Of Interrupt signal to the interrupt controller, if it is required by the controller. Each interrupt controller has its own BOI and EOI routine. They are located in the BSP, and their address and parameter are taken by the **intEoiGet** function pointer (set to **sysIntEoiGet()** in the BSP). The Tornado 2, and later, BSPs should use the BOI and EOI mechanism with **intEoiGet** function pointer.

To keep the Tornado 1.0.1 BSP backward compatible, the function pointer **intEOI** is not removed. If **intEoiGet** is **NULL**, it should be set to the **sysIntEoiGet()** routine in the BSP, **intHandlerCreate()** and the **intEOI** function pointer (set to **sysIntEOI()** in the Tornado 101 BSP) is used.

**RETURNS** A pointer to the new interrupt handler, or **NULL** if memory is insufficient.

**SEE ALSO** **intArchLib**

---

## intLevelSet()

**NAME** **intLevelSet()** – set the interrupt level (68K, x86, ARM, SimSolaris, SimNT and SH)

**SYNOPSIS**

```
int intLevelSet
(
    int level                /* new interrupt level mask */
)
```

**DESCRIPTION** This routine changes the interrupt mask in the status register to take on the value specified by *level*. Interrupts are locked out at or below that level. The value of *level* must be in the following range:

|             |                               |
|-------------|-------------------------------|
| MC680x0:    | 0 - 7                         |
| SH:         | 0 - 15                        |
| ARM:        | BSP-specific                  |
| SimSolaris: | 0 - 1                         |
| x86:        | interrupt controller specific |

On x86 systems, there are no interrupt level in the processor and the external interrupt controller manages the interrupt level. Therefore this routine does nothing and returns **OK** always.

---

**NOTE:** With the NT simulator, this routine does nothing.

---

**WARNING:** Do not call VxWorks system routines with interrupts locked. Violating this rule may re-enable interrupts unpredictably.

---

**RETURNS** The previous interrupt level.

**SEE ALSO** **intArchLib**

---

## intLock()

**NAME** **intLock()** – lock out interrupts

**SYNOPSIS** **int intLock (void)**

**DESCRIPTION** This routine disables interrupts. The **intLock()** routine returns an architecture-dependent lock-out key representing the interrupt level prior to the call; this key can be passed to **intUnlock()** to re-enable interrupts.

For MC680x0, x86, and SH architectures, interrupts are disabled at the level set by **intLockLevelSet()**. The default lock-out level is the highest interrupt level (MC680x0 = 7, x86 = 1, SH = 15).

For SimSolaris architecture, interrupts are masked. Lock-out level returned is 1 if interrupts were already locked, 0 otherwise.

For SimNT, a windows semaphore is used to lock the interrupts. Lock-out level returned is 1 if interrupts were already locked, 0 otherwise.

For MIPS processors, interrupts are disabled at the master lock-out level; this means no interrupt can occur even if unmasked in the *IntMask* bits (15-8) of the status register.

For ARM processors, interrupts (IRQs) are disabled by setting the I bit in the CPSR. This means no IRQs can occur.

For PowerPC processors, there is only one interrupt vector. The external interrupt (vector offset 0x500) is disabled when **intLock()** is called; this means that the processor cannot be interrupted by any external event.

**IMPLEMENTATION** The lock-out key is implemented differently for different architectures:

**intLock()**

|             |                                                     |
|-------------|-----------------------------------------------------|
| MC680x0:    | interrupt field mask                                |
| MIPS:       | status register                                     |
| x86:        | interrupt enable flag (IF) bit from EFLAGS register |
| PowerPC:    | MSR register value                                  |
| RM          | I bit from the CPSR                                 |
| H:          | status register                                     |
| SimSolaris: | 1 or 0                                              |
| IMNT:       | 1 or 0                                              |

---

**WARNING:** Do not call VxWorks system routines with interrupts locked. Violating this rule may re-enable interrupts unpredictably.

---

The routine **intLock()** can be called from either interrupt or task level. When called from a task context, the interrupt lock level is part of the task context. Locking out interrupts does not prevent rescheduling. Thus, if a task locks out interrupts and invokes kernel services that cause the task to block (*e.g.*, **taskSuspend()** or **taskDelay()**) or that cause a higher priority task to be ready (*e.g.*, **semGive()** or **taskResume()**), then rescheduling occurs and interrupts are unlocked while other tasks run. Rescheduling may be explicitly disabled with **taskLock()**. Traps must be enabled when calling this routine.

**EXAMPLES**

```
lockKey = intLock ();
... (work with interrupts locked out)
intUnlock (lockKey);
```

To lock out interrupts and task scheduling as well (see WARNING above):

```
if (taskLock() == OK)
{
    lockKey = intLock ();
    ... (critical section)
    intUnlock (lockKey);
    taskUnlock();
}
else
{
    ... (error message or recovery attempt)
}
```

**RETURNS**

An architecture-dependent lock-out key for the interrupt level prior to the call.

**SEE ALSO**

**intArchLib**, **intUnlock()**, **taskLock()**, **intLockLevelSet()**

---

## intLockLevelGet()

**NAME** `intLockLevelGet()` – get the current interrupt lock-out level (68K, x86, ARM, SH, SimSolaris, SimNT)

**SYNOPSIS** `int intLockLevelGet (void)`

**DESCRIPTION** This routine returns the current interrupt lock-out level, which is set by `intLockLevelSet()` and stored in the globally accessible variable `intLockMask`. This is the interrupt level currently masked when interrupts are locked out by `intLock()`. The default lock-out level (MC680x0 = 7, x86 = 1, SH = 15) is initially set by `kernelInit()` when VxWorks is initialized.

---

**NOTE:** With the NT simulator, this routine does nothing.

---

**RETURNS** The interrupt level currently stored in the interrupt lock-out mask. (ARM = **ERROR** always)

**SEE ALSO** `intArchLib`, `intLockLevelSet()`

---

## intLockLevelSet()

**NAME** `intLockLevelSet()` – set the current interrupt lock-out level (68K, x86, ARM, SH, SimSolaris, SimNT)

**SYNOPSIS**

```
void intLockLevelSet
(
    int newLevel          /* new interrupt level */
)
```

**DESCRIPTION** This routine sets the current interrupt lock-out level and stores it in the globally accessible variable `intLockMask`. The specified interrupt level is masked when interrupts are locked by `intLock()`. The default lock-out level (MC680x0 = 7, x86 = 1, SH = 15) is initially set by `kernelInit()` when VxWorks is initialized.

---

**NOTE:** With SimSolaris and SimNT, this routine does nothing.

---

**NOTE:** On the ARM, this call establishes the interrupt level to be set when `intLock()` is called.

---

**RETURNS** N/A

**SEE ALSO** `intArchLib`, `intLockLevelGet()`, `intLock()`, `taskLock()`

---

## **intSRGet()**

**NAME** `intSRGet()` – read the contents of the status register (MIPS)

**SYNOPSIS** `int intSRGet (void)`

**DESCRIPTION** This routine reads and returns the contents of the MIPS status register.

**RETURNS** The previous contents of the status register.

**SEE ALSO** `intArchLib`

---

## **intSRSet()**

**NAME** `intSRSet()` – update the contents of the status register (MIPS)

**SYNOPSIS**

```
int intSRSet
(
    int value          /* value to write to status register */
)
```

**DESCRIPTION** This routine updates and returns the previous contents of the MIPS status register.

**RETURNS** The previous contents of the status register.

**SEE ALSO** `intArchLib`

---

## intStackEnable()

**NAME** `intStackEnable()` – enable or disable the interrupt stack usage (x86)

**SYNOPSIS**

```
STATUS intStackEnable  
(  
    BOOL enable           /* TRUE to enable, FALSE to disable */  
)
```

**DESCRIPTION** This routine enables or disables the interrupt stack usage and is only callable from the task level. An Error is returned for any other calling context. The interrupt stack usage is disabled in the default configuration for the backward compatibility. Routines that manipulate the interrupt stack, are located in the file `i86/windALib.s`. These routines include `intStackEnable()`, `intEnt()` and `intExit()`.

**RETURNS** OK, or ERROR if it is not in the task level.

**SEE ALSO** `intArchLib`

---

## intUninitVecSet()

**NAME** `intUninitVecSet()` – set the uninitialized vector handler (ARM)

**SYNOPSIS**

```
void intUninitVecSet  
(  
    VOIDFUNCPTR routine    /* ptr to user routine */  
)
```

**DESCRIPTION** This routine installs a handler for the uninitialized vectors to be called when any uninitialized vector is entered.

**RETURNS** N/A.

**SEE ALSO** `intArchLib`

---

## intUnlock()

|                    |                                                                                                                                                                                                        |
|--------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>        | <b>intUnlock()</b> – cancel interrupt locks                                                                                                                                                            |
| <b>SYNOPSIS</b>    | <pre>void intUnlock (     int lockKey          /* lock-out key returned by preceding intLock() */ )</pre>                                                                                              |
| <b>DESCRIPTION</b> | This routine re-enables interrupts that have been disabled by <b>intLock()</b> . The parameter <i>lockKey</i> is an architecture-dependent lock-out key returned by a preceding <b>intLock()</b> call. |
| <b>RETURNS</b>     | N/A                                                                                                                                                                                                    |
| <b>SEE ALSO</b>    | <b>intArchLib</b> , <b>intLock()</b>                                                                                                                                                                   |

---

## intVecBaseGet()

|                    |                                                                                                                |
|--------------------|----------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>        | <b>intVecBaseGet()</b> – get the vector (trap) base address (68K, x86, MIPS, ARM, SimSolaris, SimNT)           |
| <b>SYNOPSIS</b>    | <pre>FUNCPTR *intVecBaseGet (void)</pre>                                                                       |
| <b>DESCRIPTION</b> | This routine returns the current vector base address, which is set with <b>intVecBaseSet()</b> .               |
| <b>RETURNS</b>     | The current vector base address (MIPS = 0 always, ARM = 0 always, SimSolaris = 0 always and SimNT = 0 always). |
| <b>SEE ALSO</b>    | <b>intArchLib</b> , <b>intVecBaseSet()</b>                                                                     |

---

## intVecBaseSet()

- NAME** `intVecBaseSet()` – set the vector (trap) base address (68K, x86, MIPS, ARM, SimSolaris, SimNT)
- SYNOPSIS**
- ```
void intVecBaseSet
(
    FUNCPTR * baseAddr    /* new vector (trap) base address */
)
```
- DESCRIPTION** This routine sets the vector (trap) base address. The CPU's vector base register is set to the specified value, and subsequent calls to `intVecGet()` or `intVecSet()` will use this base address. The vector base address is initially 0, until modified by calls to this routine.
- NOTE 68000** The 68000 has no vector base register; thus, this routine is a no-op for 68000 systems.
- NOTE MIPS** The MIPS processors have no vector base register; thus this routine is a no-op for this architecture.
- NOTE SH77XX** This routine sets *baseAddr* to *vbr*, then loads an interrupt dispatch code to (*vbr* + 0x600). When SH77XX processor accepts an interrupt request, it sets an exception code to INTEVT register and jumps to (*vbr* + 0x600). Thus this dispatch code is commonly used for all interrupts' handling.
- The exception codes are 12bits width, and interleaved by 0x20. VxWorks for SH77XX locates a vector table at (*vbr* + 0x800), and defines the vector offsets as (exception codes / 8). This vector table is commonly used by all interrupts, exceptions, and software traps.
- All SH77XX processors have INTEVT register at address 0xfffffd8. The SH7707 processor has yet another INTEVT2 register at address 0x04000000, to identify its enhanced interrupt sources. The dispatch code obtains the address of INTEVT register from a global constant `intEvtAdrs`. The constant is defined in `sysLib`, thus the selection of INTEVT/INTEVT2 is configurable at BSP level. The `intEvtAdrs` is loaded to (*vbr* + 4) by `intVecBaseSet()`.
- After fetching the exception code, the interrupt dispatch code applies a new interrupt mask to the status register, and jumps to an individual interrupt handler. The new interrupt mask is taken from `intPrioTable[]`, which is defined in `sysALib`. The `intPrioTable[]` is loaded to (*vbr* + 0xc00) by `intVecBaseSet()`.
- NOTE ARM** The ARM processors have no vector base register; thus this routine is a no-op for this architecture.
- NOTE SIMSOLARIS, SIMNT** This routine does nothing.

**RETURNS** N/A

**SEE ALSO** **intArchLib**, **intVecBaseGet()**, **intVecGet()**, **intVecSet()**

---

## intVecGet()

**NAME** **intVecGet()** – get an interrupt vector (68K, x86, MIPS, SH, SimSolaris, SimNT)

**SYNOPSIS**

```
FUNCPTR intVecGet  
(  
    FUNCPTR * vector          /* vector offset */  
)
```

**DESCRIPTION** This routine returns a pointer to the exception/interrupt handler attached to a specified vector. The vector is specified as an offset into the CPU's vector table. This vector table starts, by default, at:

C680x0:	0
MIPS:	<b>excBsrTbl</b> in <b>excArchLib</b>
86:	0
SH702x/SH703x/SH704x/SH76xx:	<b>excBsrTbl</b> in <b>excArchLib</b>
SH77xx:	vbr + 0x800
SimSolaris:	0

However, the vector table may be set to start at any address with **intVecBaseSet()** (on CPUs for which it is available).

This routine takes an interrupt vector as a parameter, which is the byte offset into the vector table. Macros are provided to convert between interrupt vectors and interrupt numbers, see **intArchLib**.

**NOTE SIMNT** This routine does nothing and always returns 0.

**RETURNS** A pointer to the exception/interrupt handler attached to the specified vector.

**SEE ALSO** **intArchLib**, **intVecSet()**, **intVecBaseSet()**

---

## intVecGet2()

**NAME** `intVecGet2()` – get a CPU vector, gate type(int/trap), and gate selector (x86)

**SYNOPSIS**

```
void intVecGet2
(
    FUNCPTR * vector,          /* vector offset */
    FUNCPTR * pFunction,      /* address to place in vector */
    int *    pIdtGate,        /* IDT_TRAP_GATE or IDT_INT_GATE */
    int *    pIdtSelector    /* sysCsExc or sysCsInt */
)
```

**DESCRIPTION** This routine gets a pointer to the exception/interrupt handler attached to a specified vector, the type of the gate, the selector of the gate. The vector is specified as an offset into the CPU's vector table. This vector table starts, by default, at address 0. However, the vector table may be set to start at any address with `intVecBaseSet()`.

**RETURNS** N/A

**SEE ALSO** `intArchLib`, `intVecBaseSet()`, `intVecGet()`, `intVecSet()`, `intVecSet2()`

---

## intVecSet()

**NAME** `intVecSet()` – set a CPU vector (trap) (68K, x86, MIPS, SH, SimSolaris, SimNT)

**SYNOPSIS**

```
void intVecSet
(
    FUNCPTR * vector,          /* vector offset */
    FUNCPTR  function         /* address to place in vector */
)
```

**DESCRIPTION** This routine attaches an exception/interrupt/trap handler to a vector. The vector is specified as an offset into the CPU's vector table. By default the vector table starts at:

MC680x0:	0
MIPS:	<code>excBsrTbl</code> in <code>excArchLib</code>
x86:	0
SH702x/SH703x/SH704x/SH76xx:	<code>excBsrTbl</code> in <code>excArchLib</code>
SH77xx:	<code>vbr + 0x800</code>
SimSolaris:	0

However, the vector table may be set to start at any address with **intVecBaseSet()** (on CPUs for which it is available). The vector table is set up in **usrInit()**.

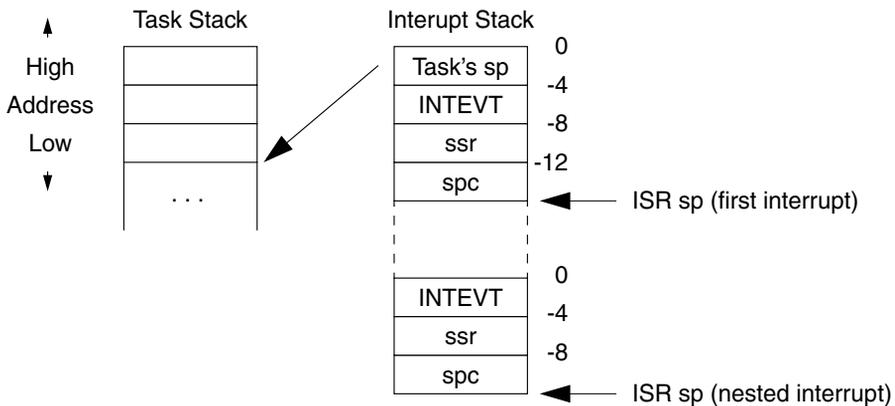
This routine takes an interrupt vector as a parameter, which is the byte offset into the vector table. Macros are provided to convert between interrupt vectors and interrupt numbers, see **intArchLib**.

**NOTE MIPS**

On MIPS CPUs the vector table is set up statically in software.

**NOTE SH77XX**

The specified interrupt handler *function* has to coordinate with an interrupt stack frame which is specially designed for the SH77XX version of VxWorks:



This interrupt stack frame is formed by a common interrupt dispatch code which is loaded at (vbr + 0x600). You usually do not have to pay any attention to this stack frame, since **intConnect()** automatically appends an appropriate stack manipulation code to your interrupt service routine. The **intConnect()** assumes that your interrupt service routine (ISR) is written in C, thus it also wraps your ISR in minimal register save/restore codes. However if you need a very fast response time to a particular interrupt request, you might want to skip this register save/restore sequence by directly attaching your ISR to the corresponding vector table entry using **intVecSet()**. Note that this technique is only applicable to an interrupt service with NO VxWorks system call. For example it is not allowed to use **semGive()** or **logMsg()** in the interrupt service routine which is directly attached to vector table by **intVecSet()**. To facilitate the direct usage of **intVecSet()** by user, a special entry point to exit an interrupt context is provided within the SH77XX version of VxWorks kernel. This entry point is located at address (vbr + intRte1W), here the intRte1W is a global symbol for the vbr offset of the entry point in 16 bit length. This entry point **intRte1** assumes that the current register bank is 0 (SR.RB == 0), and r1 and r0 are still saved on the interrupt stack, and it also requires 0x70000000 in r0. Then **intRte1** properly cleans up the interrupt stack and executes *rte* instruction to return to the previous interrupt or task context. The following code is an example of **intRte1** usage.

Here the corresponding `intPrioTable[]` entry is assumed to be `0x400000X0`, namely `MD=1`, `RB=0`, `BL=0` at the beginning of `usrIsr1`.

```

        .text
        .align 2
        .global _usrIsr1
        .type _usrIsr1,@function
        .extern _usrRtn
        .extern intRte1W

/* intPrioTable[] sets SR to 0x400000X0 */
_usrIsr1:
    mov.l r0,@-sp /* must save r0 first (BANK0) */
    mov.l r1,@-sp /* must save r1 second (BANK0) */
    mov.l r2,@-sp /* save rest of volatile registers (BANK0) */
    mov.l r3,@-sp
    mov.l r4,@-sp
    mov.l r5,@-sp
    mov.l r6,@-sp
    mov.l r7,@-sp
    sts.l pr,@-sp
    sts.l mach,@-sp
    sts.l macl,@-sp
    mov.l UsrRtn,r0
    jsr @r0 /* call user's C routine */
    nop /* (delay slot) */
    lds.l @sp+,macl /* restore volatile registers (BANK0) */
    lds.l @sp+,mach
    lds.l @sp+,pr
    mov.l @sp+,r7
    mov.l @sp+,r6
    mov.l @sp+,r5
    mov.l @sp+,r4
    mov.l @sp+,r3
    mov.l @sp+,r2

/* intRte1 restores r1 and r0 */
    mov.l IntRte1W,r1
    mov.w @r1,r0
    stc vbr,r1
    add r0,r1
    mov.l IntRteSR,r0 /* r0: 0x70000000 */
    jmp @r1 /* let intRte1 clean up stack, then rte */
    nop /* (delay slot) */

        .align 2
UsrRtn: .long _usrRtn /* user's C routine */
IntRteSR: .long 0x70000000 /* MD=1, RB=1, BL=1 */
IntRte1W: .long intRte1W

```

**intVecSet()**

The **intRte1** sets r0 to status register (SR: 0x70000000), to safely restore SPC/SSR and to clean up the interrupt stack. Note that TLB mis-hit exception immediately reboots CPU while SR.BL=1. To avoid this fatal condition, VxWorks loads the **intRte1** code and the interrupt stack to a physical address space (P1) where no TLB mis-hit happens.

Furthermore, there is another special entry point called **intRte2** at an address (vbr + intRte2W). The **intRte2** assumes that SR is already set to 0x70000000 (MD: 1, RB: 1, BL: 1), then it does not restore r1 and r0. While SR value is 0x70000000, you may use r0,r1,r2,r3 in BANK1 as volatile registers. The rest of BANK1 registers (r4,r5,r6,r7) are non-volatile, so if you need to use them then you have to preserve their original values by saving/restoring them on the interrupt stack. So, if you need the ultimate interrupt response time, you may set the corresponding **intPrioTable[]** entry to NULL and manage your interrupt service only with r0,r1,r2,r3 in BANK1 as shown in the next sample code:

```
.text
.global _usrIsr2
.type _usrIsr2,@function
.extern _usrIntCnt /* interrupt counter */
.extern intRte2W
.align 2

/* MD=1, RB=1, BL=1, since SR is not */
/* substituted from intPrioTable[]. */
_usrIsr2:
    mov.l  UsrIntAck,r1
    mov    #0x1,r0
    mov.b  r0,@r1 /* acknowledge interrupt */
    mov.l  UsrIntCnt,r1
    mov.l  X1FFFFFF,r2
    mov.l  X80000000,r3
    and    r2,r1
    or     r3,r1 /* r1: _usrIntCnt address in P1 */
    mov.l  @r1,r0
    add    #1,r0
    mov.l  r0,@r1 /* increment counter */
    mov.l  IntRte2W,r1
    and    r2,r1
    or     r3,r1 /* r1: intRte2W address in P1 */
    mov.w  @r1,r0
    stc    vbr,r1
    add    r1,r0
    jmp    @r0 /* let intRte2 clean up stack, then rte */
    nop    /* (delay slot) */

    .align 2
UsrIntAck: .long 0xa0001234 /* interrupt acknowledge register */
UsrIntCnt: .long _usrIntCnt
IntRte2W: .long intRte2W
X1FFFFFF: .long 0x1fffffff
```

```
x80000000:    .long    0x80000000
```

Note that the entire interrupt service is executed under SR.BL=1 in this sample code. It means that any access to virtual address space may reboot CPU, since TLB mis-hit exception is blocked. Therefore `usrIsr2` has to access `usrIntCnt` and `intRte2W` from P1 region. Also `usrIsr2` itself has to be executed on P1 region, and it can be done by relocating the address of `usrIsr2` to P1 as shown below:

```
IMPORT void usrIsr2 (void);
intVecSet (vector, (FUNCPTR)((UINT32) usrIsr2 & 0x1fffffff | 0x80000000));
```

In conclusion, you have to guarantee that the entire ISR does not access to any virtual address space if you set the corresponding `intPrioTable[]` entry to NULL.

<b>NOTE SIMNT</b>	This routine does nothing.
<b>RETURNS</b>	N/A
<b>SEE ALSO</b>	<code>intArchLib</code> , <code>intVecBaseSet()</code> , <code>intVecGet()</code>

---

## intVecSet2()

<b>NAME</b>	<code>intVecSet2()</code> – set a CPU vector, gate type(int/trap), and selector (x86)
<b>SYNOPSIS</b>	<pre><b>void intVecSet2</b> <b>(</b>     <b>FUNCPTR * vector,</b>           <b>/* vector offset */</b>     <b>FUNCPTR  function,</b>       <b>/* address to place in vector */</b>     <b>int      idtGate,</b>         <b>/* IDT_TRAP_GATE or IDT_INT_GATE */</b>     <b>int      idtSelector</b>      <b>/* sysCsExc or sysCsInt */</b> <b>)</b></pre>
<b>DESCRIPTION</b>	This routine attaches an exception handler to a specified vector, with the type of the gate and the selector of the gate. The vector is specified as an offset into the CPU's vector table. This vector table starts, by default, at address 0. However, the vector table may be set to start at any address with <code>intVecBaseSet()</code> . The vector table is set up in <code>usrInit()</code> .
<b>RETURNS</b>	N/A
<b>SEE ALSO</b>	<code>intArchLib</code> , <code>intVecBaseSet()</code> , <code>intVecGet()</code> , <code>intVecSet()</code> , <code>intVecGet2()</code>

---

## intVecTableWriteProtect()

<b>NAME</b>	<code>intVecTableWriteProtect()</code> – write-protect exception vector table (68K, x86, ARM, SimSolaris, SimNT)
<b>SYNOPSIS</b>	<b>STATUS</b> <code>intVecTableWriteProtect (void)</code>
<b>DESCRIPTION</b>	<p>If the unbundled Memory Management Unit (MMU) support package (VxVMI) is present, this routine write-protects the exception vector table to protect it from being accidentally corrupted.</p> <p>Note that other data structures contained in the page will also be write-protected. In the default VxWorks configuration, the exception vector table is located at location 0 in memory. Write-protecting this affects the backplane anchor, boot configuration information, and potentially the text segment (assuming the default text location of 0x1000.) All code that manipulates these structures has been modified to write-enable memory for the duration of the operation. If you select a different address for the exception vector table, be sure it resides in a page separate from other writable data structures.</p> <hr/> <p><b>NOTE:</b> This routine always returns <b>ERROR</b> on simulators.</p> <hr/>
<b>RETURNS</b>	OK, or <b>ERROR</b> if memory cannot be write-protected.
<b>ERRNO</b>	<code>S_intLib_VEC_TABLE_WP_UNAVAILABLE</code>
<b>SEE ALSO</b>	<code>intArchLib</code>

---

## ioctl()

<b>NAME</b>	<code>ioctl()</code> – perform an I/O control function
<b>SYNOPSIS</b>	<pre>int ioctl (     int fd,                /* file descriptor */     int function,          /* function code */     int arg                 /* arbitrary argument */ )</pre>

**DESCRIPTION** This routine performs an I/O control function on a device. The control functions used by VxWorks device drivers are defined in the header file **ioLib.h**. Most requests are passed on to the driver for handling. Since the availability of **ioctl()** functions is driver-specific, these functions are discussed separately in **tyLib**, **pipeDrv**, **nfsDrv**, **dosFsLib**, **rt11FsLib**, and **rawFsLib**.

The following example renames the file or directory to the string “newname”:

```
ioctl (fd, FIORENAME, "newname");
```

Note that the function **FIOGETNAME** is handled by the I/O interface level and is not passed on to the device driver itself. Thus this function code value should not be used by customer-written drivers.

**RETURNS** The return value of the driver, or **ERROR** if the file descriptor does not exist.

**SEE ALSO** **ioLib**, **tyLib**, **pipeDrv**, **nfsDrv**, **dosFsLib**, **rt11FsLib**, **rawFsLib**, *VxWorks Programmer's Guide: I/O System, Local File Systems*

---

## ioDefPathGet()

**NAME** **ioDefPathGet()** – get the current default path

**SYNOPSIS**

```
void ioDefPathGet  
(  
    char * pathname          /* where to return the name */  
)
```

**DESCRIPTION** This routine copies the name of the current default path to *pathname*. The parameter *pathname* should be **MAX\_FILENAME\_LENGTH** characters long.

**RETURNS** N/A

**SEE ALSO** **ioLib**, **ioDefPathSet()**, **chdir()**, **getcwd()**

---

## ioDefPathSet()

<b>NAME</b>	<b>ioDefPathSet()</b> – set the current default path
<b>SYNOPSIS</b>	<pre><b>STATUS</b> ioDefPathSet (     <b>char * name</b>                /* name of the new default device and path */ )</pre>
<b>DESCRIPTION</b>	This routine sets the default I/O path. All relative pathnames specified to the I/O system will be prepended with this pathname. This pathname must be an absolute pathname, <i>i.e.</i> , <i>name</i> must begin with an existing device name.
<b>RETURNS</b>	OK, or ERROR if the first component of the pathname is not an existing device.
<b>SEE ALSO</b>	<b>ioLib</b> , <b>ioDefPathGet()</b> , <b>chdir()</b> , <b>getcwd()</b>

---

## ioGlobalStdGet()

<b>NAME</b>	<b>ioGlobalStdGet()</b> – get the file descriptor for global standard input/output/error
<b>SYNOPSIS</b>	<pre><b>int</b> ioGlobalStdGet (     <b>int stdFd</b>                /* std input (0), output (1), or error (2) */ )</pre>
<b>DESCRIPTION</b>	This routine returns the current underlying file descriptor for global standard input, output, and error.
<b>RETURNS</b>	The underlying global file descriptor, or ERROR if <i>stdFd</i> is not 0, 1, or 2.
<b>SEE ALSO</b>	<b>ioLib</b> , <b>ioGlobalStdSet()</b> , <b>ioTaskStdGet()</b>

---

## ioGlobalStdSet()

<b>NAME</b>	<code>ioGlobalStdSet()</code> – set the file descriptor for global standard input/output/error
<b>SYNOPSIS</b>	<pre>void ioGlobalStdSet (     int stdFd,          /* std input (0), output (1), or error (2) */     int newFd          /* new underlying file descriptor */ )</pre>
<b>DESCRIPTION</b>	This routine changes the assignment of a specified global standard file descriptor <i>stdFd</i> (0, 1, or 2) to the specified underlying file descriptor <i>newFd</i> . <i>newFd</i> should be a file descriptor open to the desired device or file. All tasks will use this new assignment when doing I/O to <i>stdFd</i> , unless they have specified a task-specific standard file descriptor (see <code>ioTaskStdSet()</code> ). If <i>stdFd</i> is not 0, 1, or 2, this routine has no effect.
<b>RETURNS</b>	N/A
<b>SEE ALSO</b>	<code>ioLib</code> , <code>ioGlobalStdGet()</code> , <code>ioTaskStdSet()</code>

---

## ioHelp()

<b>NAME</b>	<code>ioHelp()</code> – print a synopsis of I/O utility functions
<b>SYNOPSIS</b>	<pre>void ioHelp (void)</pre>
<b>DESCRIPTION</b>	This function prints out synopsis for the I/O and File System utility functions.
<b>RETURNS</b>	N/A
<b>SEE ALSO</b>	<code>usrFsLib</code> , <i>VxWorks Programmer's Guide: Target Shell</i>

---

## iosDevAdd()

**NAME** `iosDevAdd()` – add a device to the I/O system

**SYNOPSIS**

```
STATUS iosDevAdd
(
    DEV_HDR * pDevHdr, /* pointer to device's structure */
    char *    name,    /* name of device */
    int      drvnum   /* # of servicing driver, ret'd by iosDrvInstall() */
)
```

**DESCRIPTION** This routine adds a device to the I/O system device list, making the device available for subsequent `open()` and `creat()` calls.

The parameter *pDevHdr* is a pointer to a device header, `DEV_HDR` (defined in `iosLib.h`), which is used as the node in the device list. Usually this is the first item in a larger device structure for the specific device type. The parameters *name* and *drvnum* are entered in *pDevHdr*.

**RETURNS** `OK`, or `ERROR` if there is already a device with the specified name.

**SEE ALSO** `iosLib`

---

## iosDevDelete()

**NAME** `iosDevDelete()` – delete a device from the I/O system

**SYNOPSIS**

```
void iosDevDelete
(
    DEV_HDR * pDevHdr /* pointer to device's structure */
)
```

**DESCRIPTION** This routine deletes a device from the I/O system device list, making it unavailable to subsequent `open()` or `creat()` calls. No interaction with the driver occurs, and any file descriptors open on the device or pending operations are unaffected.

If the device was never added to the device list, unpredictable results may occur.

**RETURNS** `N/A`

**SEE ALSO** `iosLib`

---

## iosDevFind()

**NAME** iosDevFind() – find an I/O device in the device list

**SYNOPSIS**

```
DEV_HDR *iosDevFind
(
    char * name,           /* name of the device */
    char * *pNameTail     /* where to put ptr to tail of name */
)
```

**DESCRIPTION** This routine searches the device list for a device whose name matches the first portion of *name*. If a device is found, **iosDevFind()** sets the character pointer pointed to by *pNameTail* to point to the first character in *name*, following the portion which matched the device name. It then returns a pointer to the device. If the routine fails, it returns a pointer to the default device (that is, the device where the current working directory is mounted) and sets *pNameTail* to point to the beginning of *name*. If there is no default device, **iosDevFind()** returns NULL.

**RETURNS** A pointer to the device header, or NULL if the device is not found.

**SEE ALSO** iosLib

---

## iosDevShow()

**NAME** iosDevShow() – display the list of devices in the system

**SYNOPSIS** void iosDevShow (void)

**DESCRIPTION** This routine displays a list of all devices in the device list.

**RETURNS** N/A

**SEE ALSO** iosShow, devs(), *VxWorks Programmer's Guide: I/O System*, windsh, *Tornado User's Guide: Shell*

---

## iosDrvInstall()

**NAME** iosDrvInstall() – install an I/O driver

**SYNOPSIS**

```
int iosDrvInstall
(
    FUNCPTR pCreate,          /* pointer to driver create function */
    FUNCPTR pDelete,         /* pointer to driver delete function */
    FUNCPTR pOpen,           /* pointer to driver open function */
    FUNCPTR pClose,          /* pointer to driver close function */
    FUNCPTR pRead,           /* pointer to driver read function */
    FUNCPTR pWrite,          /* pointer to driver write function */
    FUNCPTR pIoctl           /* pointer to driver ioctl function */
)
```

**DESCRIPTION** This routine should be called once by each I/O driver. It hooks up the various I/O service calls to the driver service routines, assigns the driver a number, and adds the driver to the driver table.

**RETURNS** The driver number of the new driver, or **ERROR** if there is no room for the driver.

**SEE ALSO** iosLib

---

## iosDrvRemove()

**NAME** iosDrvRemove() – remove an I/O driver

**SYNOPSIS**

```
STATUS iosDrvRemove
(
    int drvnum,              /* no. of driver to remove, returned by */
                             /* iosDrvInstall() */
    BOOL forceClose          /* if TRUE, force closure of open files */
)
```

**DESCRIPTION** This routine removes an I/O driver (added by **iosDrvInstall()**) from the driver table.

**RETURNS** OK, or **ERROR** if the driver has open files.

**SEE ALSO** iosLib, iosDrvInstall()

---

## iosDrvShow()

<b>NAME</b>	<code>iosDrvShow()</code> – display a list of system drivers
<b>SYNOPSIS</b>	<pre>void iosDrvShow (void)</pre>
<b>DESCRIPTION</b>	This routine displays a list of all drivers in the driver list.
<b>RETURNS</b>	N/A
<b>SEE ALSO</b>	<code>iosShow</code> , <i>VxWorks Programmer's Guide: I/O System</i> , <code>windsh</code> , <i>Tornado User's Guide: Shell</i>

---

## iosFdShow()

<b>NAME</b>	<code>iosFdShow()</code> – display a list of file descriptor names in the system
<b>SYNOPSIS</b>	<pre>void iosFdShow (void)</pre>
<b>DESCRIPTION</b>	This routine displays a list of all file descriptors in the system.
<b>RETURNS</b>	N/A
<b>SEE ALSO</b>	<code>iosShow</code> , <code>ioctl()</code> , <i>VxWorks Programmer's Guide: I/O System</i> , <code>windsh</code> , <i>Tornado User's Guide: Shell</i>

---

## iosFdValue()

<b>NAME</b>	<code>iosFdValue()</code> – validate an open file descriptor and return the driver-specific value
<b>SYNOPSIS</b>	<pre>int iosFdValue (     int fd                /* file descriptor to check */ )</pre>
<b>DESCRIPTION</b>	This routine checks to see if a file descriptor is valid and returns the driver-specific value.

**RETURNS** The driver-specific value, or **ERROR** if the file descriptor is invalid.

**SEE ALSO** [iosLib](#)

---

## iosInit()

**NAME** `iosInit()` – initialize the I/O system

**SYNOPSIS**

```
STATUS iosInit
(
    int    max_drivers,      /* maximum number of drivers allowed */
    int    max_files,       /* max number of files allowed open at once */
    char * nullDevName     /* name of the null device (bit bucket) */
)
```

**DESCRIPTION** This routine initializes the I/O system. It must be called before any other I/O system routine.

**RETURNS** **OK**, or **ERROR** if memory is insufficient.

**SEE ALSO** [iosLib](#)

---

## iosShowInit()

**NAME** `iosShowInit()` – initialize the I/O system show facility

**SYNOPSIS** `void iosShowInit (void)`

**DESCRIPTION** This routine links the I/O system show facility into the VxWorks system. It is called automatically when `INCLUDE_SHOW_ROUTINES` is defined in `configAll.h`.

**RETURNS** N/A

**SEE ALSO** [iosShow](#)

---

## ioTaskStdGet()

- NAME** `ioTaskStdGet()` – get the file descriptor for task standard input/output/error
- SYNOPSIS**
- ```
int ioTaskStdGet
(
    int taskId,          /* ID of desired task (0 = self) */
    int stdFd           /* std input (0), output (1), or error (2) */
)
```
- DESCRIPTION** This routine returns the current underlying file descriptor for task-specific standard input, output, and error.
- RETURNS** The underlying file descriptor, or **ERROR** if *stdFd* is not 0, 1, or 2, or the routine is called at interrupt level.
- SEE ALSO** `ioLib`, `ioGlobalStdGet()`, `ioTaskStdSet()`

---

## ioTaskStdSet()

- NAME** `ioTaskStdSet()` – set the file descriptor for task standard input/output/error
- SYNOPSIS**
- ```
void ioTaskStdSet
(
    int taskId,          /* task whose std fd is to be set (0 = self) */
    int stdFd,          /* std input (0), output (1), or error (2) */
    int newFd           /* new underlying file descriptor */
)
```
- DESCRIPTION** This routine changes the assignment of a specified task-specific standard file descriptor *stdFd* (0, 1, or 2) to the specified underlying file descriptor *newFd*. *newFd* should be a file descriptor open to the desired device or file. The calling task will use this new assignment when doing I/O to *stdFd*, instead of the system-wide global assignment which is used by default. If *stdFd* is not 0, 1, or 2, this routine has no effect.
- 
- NOTE:** This routine has no effect if it is called at interrupt level.
- 
- RETURNS** N/A
- SEE ALSO** `ioLib`, `ioGlobalStdGet()`, `ioTaskStdGet()`

---

## ipAttach()

**NAME** ipAttach() – a generic attach routine for the TCP/IP network stack

**SYNOPSIS**

```
int ipAttach
(
    int    unit,           /* Unit number */
    char * pDevice        /* Device name (i.e. ln, ei etc.). */
)
```

**DESCRIPTION** This routine takes the unit number and device name of an END or NPT driver (e.g., “ln0”, “ei0”, etc.) and attaches the IP protocol to the corresponding device. Following a successful attachment IP will begin receiving packets from the devices.

**RETURNS** OK or ERROR

**SEE ALSO** ipProto

---

## ipDetach()

**NAME** ipDetach() – a generic detach routine for the TCP/IP network stack

**SYNOPSIS**

```
STATUS ipDetach
(
    int    unit,           /* Unit number */
    char * pDevice        /* Device name (i.e. ln, ei etc.). */
)
```

**DESCRIPTION** This routine removes the TCP/IP stack from the MUX. If completed successfully, the IP protocol will no longer receive packets from the named END driver.

**RETURNS** OK or ERROR

**SEE ALSO** ipProto

---

## ipFilterHookAdd()

**NAME** `ipFilterHookAdd()` – add a routine to receive all internet protocol packets

**SYNOPSIS**

```
STATUS ipFilterHookAdd
(
    FUNCPTR ipFilterHook    /* routine to receive raw IP packets */
)
```

**DESCRIPTION** This routine adds a hook routine that will be called for every IP packet that is received. The filter hook routine should be of the form:

```
BOOL ipFilterHook
(
    struct ifnet *pIf,      /* interface that received the packet */
    struct mbuf **pPtrMbuf, /* pointer to pointer to an mbuf chain */
    struct ip **pPtrIpHdr, /* pointer to pointer to IP header */
    int ipHdrLen,         /* IP packet header length */
)
```

The hook routine should return **TRUE** if it has handled the input packet. A returned value of **TRUE** effectively consumes the packet from the viewpoint of IP, which will never see the packet. As a result, when the filter hook returns **TRUE**, it must handle the freeing of any resources associated with the packet. For example, the filter hook routine would be responsible for freeing the packet's **mbuf** chain by calling **m\_freem(\*pPtrMbuf)**.

The filter hook routine should return **FALSE** if it has not handled the packet. In response to a **FALSE**, the network stack submits the packet for normal IP processing.

Within the packet's IP header (the filter hook can obtain a pointer to the IP header by de-referencing **pPtrIpHdr**), you will find that the values in the **ip\_len** field, the **ip\_id** field, and **ip\_offset** field have been converted to the host byte order before the packet was handed to the filter hook.

### VXWORKS AE PROTECTION DOMAINS

Under VxWorks AE, you can call **ipFilterHookAdd()** from within the kernel protection domain only, and the function referenced in the *ipFilterHook* parameter must reside in the kernel protection domain. This restriction does not apply to non-AE versions of VxWorks.

**RETURNS** OK, always.

**SEE ALSO** `ipFilterLib`

## **ipFilterHookDelete()**

<b>NAME</b>	<b>ipFilterHookDelete()</b> – delete a IP filter hook routine
<b>SYNOPSIS</b>	<b>void ipFilterHookDelete (void)</b>
<b>DESCRIPTION</b>	This routine deletes an IP filter hook.
<b>RETURNS</b>	N/A
<b>SEE ALSO</b>	<b>ipFilterLib</b>

---

## **ipFilterLibInit()**

<b>NAME</b>	<b>ipFilterLibInit()</b> – initialize IP filter facility
<b>SYNOPSIS</b>	<b>void ipFilterLibInit (void)</b>
<b>DESCRIPTION</b>	This routine links the IP filter facility into the VxWorks system. These routines are included automatically if <b>INCLUDE_IP_FILTER</b> is defined.
<b>RETURNS</b>	N/A
<b>SEE ALSO</b>	<b>ipFilterLib</b>

---

## ipstatShow()

<b>NAME</b>	<code>ipstatShow()</code> – display IP statistics
<b>SYNOPSIS</b>	<pre>void ipstatShow (     BOOL zero          /* TRUE = reset statistics to 0 */ )</pre>
<b>DESCRIPTION</b>	This routine displays detailed statistics for the IP protocol.
<b>RETURNS</b>	N/A
<b>SEE ALSO</b>	<code>netShow</code>

---

## rint()

<b>NAME</b>	<code>rint()</code> – convert a double-precision value to an integer
<b>SYNOPSIS</b>	<pre>int rint (     double x /* argument */ )</pre>
<b>DESCRIPTION</b>	This routine converts a double-precision value <i>x</i> to an integer using the selected IEEE rounding direction. <hr/> <b>WARNING:</b> The rounding direction is not pre-selectable and is fixed for round-to-the-nearest. <hr/>
<b>INCLUDE FILES</b>	<code>math.h</code>
<b>RETURNS</b>	The integer representation of <i>x</i> .
<b>SEE ALSO</b>	<code>mathALib</code>

**rintf()**

---

**rintf()**

**NAME** `rintf()` – convert a single-precision value to an integer

**SYNOPSIS**

```
int rintf
(
    float x    /* argument */
)
```

**DESCRIPTION** This routine converts a single-precision value *x* to an integer using the selected IEEE rounding direction.

---

**WARNING:** The rounding direction is not pre-selectable and is fixed as round-to-the-nearest.

---

**INCLUDE FILES** `math.h`

**RETURNS** The integer representation of *x*.

**SEE ALSO** `mathALib`

---

**iround()**

**NAME** `iround()` – round a number to the nearest integer

**SYNOPSIS**

```
int iround
(
    double x   /* argument */
)
```

**DESCRIPTION** This routine rounds a double-precision value *x* to the nearest integer value.

---

**NOTE:** If *x* is spaced evenly between two integers, it returns the even integer.

---

**INCLUDE FILES** `math.h`

**RETURNS** The integer nearest to *x*.

**SEE ALSO** `mathALib`

---

## iroundf()

**NAME** `iroundf()` – round a number to the nearest integer

**SYNOPSIS**

```
int iroundf
(
    float x    /* argument */
)
```

**DESCRIPTION** This routine rounds a single-precision value *x* to the nearest integer value.

---

**NOTE:** If *x* is spaced evenly between two integers, the even integer is returned.

---

**INCLUDE FILES** `math.h`

**RETURNS** The integer nearest to *x*.

**SEE ALSO** `mathALib`

---

## isalnum()

**NAME** `isalnum()` – test whether a character is alphanumeric (ANSI)

**SYNOPSIS**

```
int isalnum
(
    int c          /* character to test */
)
```

**DESCRIPTION** This routine tests whether *c* is a character for which `isalpha()` or `isdigit()` returns true.

**INCLUDE FILES** `ctype.h`

**RETURNS** Non-zero if and only if *c* is alphanumeric.

**SEE ALSO** `ansiCtype`

## isalpha()

<b>NAME</b>	<b>isalpha()</b> – test whether a character is a letter (ANSI)
<b>SYNOPSIS</b>	<pre>int isalpha (     int c                /* character to test */ )</pre>
<b>DESCRIPTION</b>	This routine tests whether <i>c</i> is a character for which <b>isupper()</b> or <b>islower()</b> returns true.
<b>INCLUDE FILES</b>	<b>cctype.h</b>
<b>RETURNS</b>	Non-zero if and only if <i>c</i> is a letter.
<b>SEE ALSO</b>	<b>ansiCtype</b>

---

## isatty()

<b>NAME</b>	<b>isatty()</b> – return whether the underlying driver is a tty device
<b>SYNOPSIS</b>	<pre>BOOL isatty (     int fd                /* file descriptor to check */ )</pre>
<b>DESCRIPTION</b>	This routine simply invokes the <b>ioctl()</b> function <b>FIOISATTY</b> on the specified file descriptor.
<b>RETURNS</b>	<b>TRUE</b> , or <b>FALSE</b> if the driver does not indicate a tty device.
<b>SEE ALSO</b>	<b>ioLib</b>

---

## iscntrl()

**NAME** `iscntrl()` – test whether a character is a control character (ANSI)

**SYNOPSIS**

```
int iscntrl
(
    int c                /* character to test */
)
```

**DESCRIPTION** This routine tests whether *c* is a control character.

**INCLUDE FILES** `ctype.h`

**RETURNS** Non-zero if and only if *c* is a control character.

**SEE ALSO** `ansiCtype`

---

## isdigit()

**NAME** `isdigit()` – test whether a character is a decimal digit (ANSI)

**SYNOPSIS**

```
int isdigit
(
    int c                /* character to test */
)
```

**DESCRIPTION** This routine tests whether *c* is a decimal-digit character.

**INCLUDE FILES** `ctype.h`

**RETURNS** Non-zero if and only if *c* is a decimal digit.

**SEE ALSO** `ansiCtype`

## isgraph()

<b>NAME</b>	<b>isgraph()</b> – test whether a character is a printing, non-white-space character (ANSI)
<b>SYNOPSIS</b>	<pre>int isgraph (     int c                /* character to test */ )</pre>
<b>DESCRIPTION</b>	This routine returns true if <i>c</i> is a printing character, and not a character for which <b>isspace()</b> returns true.
<b>INCLUDE FILES</b>	<b>ctype.h</b>
<b>RETURNS</b>	Non-zero if and only if <i>c</i> is a printable, non-white-space character.
<b>SEE ALSO</b>	<b>ansiCtype</b> , <b>isspace()</b>

---

## islower()

<b>NAME</b>	<b>islower()</b> – test whether a character is a lower-case letter (ANSI)
<b>SYNOPSIS</b>	<pre>int islower (     int c                /* character to test */ )</pre>
<b>DESCRIPTION</b>	This routine tests whether <i>c</i> is a lower-case letter.
<b>INCLUDE FILES</b>	<b>ctype.h</b>
<b>RETURNS</b>	Non-zero if and only if <i>c</i> is a lower-case letter.
<b>SEE ALSO</b>	<b>ansiCtype</b>

---

## isprint()

<b>NAME</b>	<b>isprint()</b> – test whether a character is printable, including the space character (ANSI)
<b>SYNOPSIS</b>	<pre>int isprint (     int c                /* character to test */ )</pre>
<b>DESCRIPTION</b>	This routine returns true if <i>c</i> is a printing character or the space character.
<b>INCLUDE FILES</b>	<code>ctype.h</code>
<b>RETURNS</b>	Non-zero if and only if <i>c</i> is printable, including the space character.
<b>SEE ALSO</b>	<code>ansiCtype</code>

---

## ispunct()

<b>NAME</b>	<b>ispunct()</b> – test whether a character is punctuation (ANSI)
<b>SYNOPSIS</b>	<pre>int ispunct (     int c                /* character to test */ )</pre>
<b>DESCRIPTION</b>	This routine tests whether a character is punctuation, <i>i.e.</i> , a printing character for which neither <code>isspace()</code> nor <code>isalnum()</code> is true.
<b>INCLUDE FILES</b>	<code>ctype.h</code>
<b>RETURNS</b>	Non-zero if and only if <i>c</i> is a punctuation character.
<b>SEE ALSO</b>	<code>ansiCtype</code>

## isspace()

<b>NAME</b>	<b>isspace()</b> – test whether a character is a white-space character (ANSI)
<b>SYNOPSIS</b>	<pre>int isspace (     int c                /* character to test */ )</pre>
<b>DESCRIPTION</b>	This routine tests whether a character is a standard white-space characters, as follows: space            ‘ ‘ horizontal tab   \t vertical tab      \v carriage return  \r new-line         \n form-feed        \f
<b>INCLUDE FILES</b>	<b>ctype.h</b>
<b>RETURNS</b>	Non-zero if and only if <i>c</i> is a space, tab, carriage return, new-line, or form-feed character.
<b>SEE ALSO</b>	<b>ansiCtype</b>

---

## isupper()

<b>NAME</b>	<b>isupper()</b> – test whether a character is an upper-case letter (ANSI)
<b>SYNOPSIS</b>	<pre>int isupper (     int c                /* character to test */ )</pre>
<b>DESCRIPTION</b>	This routine tests whether <i>c</i> is an upper-case letter.
<b>INCLUDE FILES</b>	<b>ctype.h</b>
<b>RETURNS</b>	Non-zero if and only if <i>c</i> is an upper-case letter.
<b>SEE ALSO</b>	<b>ansiCtype</b>

---

## isxdigit()

**NAME** `isxdigit()` – test whether a character is a hexadecimal digit (ANSI)

**SYNOPSIS**

```
int isxdigit
(
    int c                /* character to test */
)
```

**DESCRIPTION** This routine tests whether *c* is a hexadecimal-digit character.

**INCLUDE FILES** `ctype.h`

**RETURNS** Non-zero if and only if *c* is a hexadecimal digit.

**SEE ALSO** `ansiCtype`

---

## kernelInit()

**NAME** kernelInit() – initialize the kernel

**SYNOPSIS**

```
void kernelInit
(
    FUNCPTR  rootRtn,          /* user start-up routine */
    unsigned rootMemSize,     /* memory for TCB and root stack */
    char *   pMemPoolStart,   /* beginning of memory pool */
    char *   pMemPoolEnd,     /* end of memory pool */
    unsigned intStackSize,    /* interrupt stack size */
    int      lockOutLevel     /* interrupt lock-out level (1-7) */
)
```

**DESCRIPTION** This routine initializes and starts the kernel. It should be called only once. The parameter *rootRtn* specifies the entry point of the user's start-up code that subsequently initializes system facilities (*i.e.*, the I/O system, network). Typically, *rootRtn* is set to **usrRoot()**.

Interrupts are enabled for the first time after **kernelInit()** exits. VxWorks will not exceed the specified interrupt lock-out level during any of its brief uses of interrupt locking as a means of mutual exclusion.

The system memory partition is initialized by **kernelInit()** with the size set by *pMemPoolStart* and *pMemPoolEnd*. Architectures that support a separate interrupt stack allocate a portion of memory for this purpose, of *intStackSize* bytes starting at *pMemPoolStart*.

---

**NOTE:** On SH77xx architectures, the interrupt stack is emulated by software, and it has to be located in a fixed physical address space (P1 or P2) if the on-chip MMU is enabled. If *pMemPoolStart* is in a logical address space (P0 or P3), the interrupt stack area is reserved on the same logical address space. The actual interrupt stack is relocated to a fixed physical space pointed by VBR.

---

**RETURNS** N/A

**SEE ALSO** kernelLib, intLockLevelSet()

---

## kernelTimeSlice()

**NAME** `kernelTimeSlice()` – enable round-robin selection

**SYNOPSIS**

```
STATUS kernelTimeSlice
(
    int ticks                /* time-slice in ticks or 0 to disable */
                            /* round-robin */
)
```

**DESCRIPTION** This routine enables round-robin selection among tasks of same priority and sets the system time-slice to *ticks*. Round-robin scheduling is disabled by default. A time-slice of zero ticks disables round-robin scheduling.

For more information about round-robin scheduling, see the manual entry for **kernelLib**.

**RETURNS** OK, always.

**SEE ALSO** `kernelLib`

---

## kernelVersion()

**NAME** `kernelVersion()` – return the kernel revision string

**SYNOPSIS** `char *kernelVersion (void)`

**DESCRIPTION** This routine returns a string which contains the current revision of the kernel. The string is of the form “WIND version x.y”, where “x” corresponds to the kernel major revision, and “y” corresponds to the kernel minor revision.

**RETURNS** A pointer to a string of format “WIND version x.y”.

**SEE ALSO** `kernelLib`



## kill()

<b>NAME</b>	kill() – send a signal to a task (POSIX)
<b>SYNOPSIS</b>	<pre>int kill (     int tid,                /* task to send signal to */     int signo              /* signal to send to task */ )</pre>
<b>DESCRIPTION</b>	This routine sends a signal <i>signo</i> to the task specified by <i>tid</i> .
<b>RETURNS</b>	OK (0), or ERROR (-1) if the task ID or signal number is invalid.
<b>ERRNO</b>	EINVAL
<b>SEE ALSO</b>	sigLib

---

## l()

**NAME** l() – disassemble and display a specified number of instructions

**SYNOPSIS**

```
void l
(
    INSTR * addr,          /* address of first instruction to */
                          /* disassemble if 0, continue from the last */
                          /* instruction disassembled on the last call */
                          /* to l */
    int     count         /* number of instruction to disassemble if */
                          /* 0, use the same as the last call to l */
)
```

**DESCRIPTION** This routine disassembles a specified number of instructions and displays them on standard output. If the address of an instruction is entered in the system symbol table, the symbol will be displayed as a label for that instruction. Also, addresses in the opcode field of instructions will be displayed symbolically.

To execute, enter:

```
-> l [address [,count]]
```

If *address* is omitted or zero, disassembly continues from the previous address. If *count* is omitted or zero, the last specified count is used (initially 10). As with all values entered via the shell, the address may be typed symbolically.

**RETURNS** N/A

**SEE ALSO** **dbgLib**, *VxWorks Programmer's Guide: Target Shell*, **windsh**, *Tornado User's Guide: Shell*

---

## labs()

**NAME** labs() – compute the absolute value of a long (ANSI)

**SYNOPSIS**

```
long labs
(
    long i                /* long for which to return absolute value */
)
```

**ld()**

<b>DESCRIPTION</b>	This routine computes the absolute value of a specified <b>long</b> . If the result cannot be represented, the behavior is undefined. This routine is equivalent to <b>abs()</b> , except that the argument and return value are all of type <b>long</b> .
<b>INCLUDE FILES</b>	<b>stdlib.h</b>
<b>RETURNS</b>	The absolute value of <i>i</i> .
<b>SEE ALSO</b>	<b>ansiStdlib</b>

**ld()**

<b>NAME</b>	<b>ld()</b> – load an object module into memory
<b>SYNOPSIS</b>	<pre> MODULE_ID ld (     int    syms,           /* -1, 0, or 1 */     BOOL   noAbort,       /* TRUE = don't abort script on error */     char * name           /* name of object module, NULL = standard input */ ) </pre>

**DESCRIPTION** This command loads an object module from a file or from standard input. The object module must be in UNIX **a.out** format. External references in the module are resolved during loading. The *syms* parameter determines how symbols are loaded; possible values are:

- 0 - Add global symbols to the system symbol table.
- 1 - Add global and local symbols to the system symbol table.
- 1 - Add no symbols to the system symbol table.

If there is an error during loading (*e.g.*, externals undefined, too many symbols, etc.), then **shellScriptAbort()** is called to stop any script that this routine was called from. If *noAbort* is **TRUE**, errors are noted but ignored.

The normal way of using **ld()** is to load all symbols (*syms* = 1) during debugging and to load only global symbols later.

The routine **ld()** is a **shell command**. That is, it is designed to be used only in the shell, and not in code running on the target. In future releases, calling **ld()** directly from code may not be supported.

## COMMON SYMBOLS

On the target shell, for the **ld** command only, common symbol behavior is determined by the value of the global variable, **ldCommonMatchAll**. The reasoning for **ldCommonMatchAll** matches the purpose of the **windsh** environment variable, **LD\_COMMON\_MATCH\_ALL** as explained below.

If **ldCommonMatchAll** is set to **TRUE** (equivalent to **windsh** “**LD\_COMMON\_MATCH\_ALL=on**”), the loader tries to match a common symbol with an existing one. If a symbol with the same name is already defined, the loader takes its address. Otherwise, the loader creates a new entry. If set to **FALSE** (equivalent to **windsh** “**LD\_COMMON\_MATCH\_ALL=off**”), the loader does not try to find an existing symbol. It creates an entry for each common symbol.

## EXAMPLE

The following example loads the **a.out** file **module** from the default file device into memory, and adds any global symbols to the symbol table:

```
-> ld <module
```

This example loads **test.o** with all symbols:

```
-> ld 1,0,"test.o"
```

## RETURNS

**MODULE\_ID**, or **NULL** if there are too many symbols, the object file format is invalid, or there is an error reading the file.

## SEE ALSO

**usrLib**, **loadLib**, *VxWorks Programmer's Guide: Target Shell*, **windsh**, *Tornado User's Guide: Shell*

---

## ldexp()

## NAME

**ldexp()** – multiply a number by an integral power of 2 (ANSI)

## SYNOPSIS

```
double ldexp
(
    double v,                /* a floating point number */
    int xexp                 /* exponent */
)
```

## DESCRIPTION

This routine multiplies a floating-point number by an integral power of 2. A range error may occur.

## INCLUDE FILES

**math.h**

**RETURNS** The double-precision value of *v* times 2 to the power of *xexp*.

**SEE ALSO** [ansiMath](#)

---

## ldiv()

**NAME** `ldiv()` – compute the quotient and remainder of the division (ANSI)

**SYNOPSIS**

```
ldiv_t ldiv
(
    long numer,          /* numerator */
    long denom          /* denominator */
)
```

**DESCRIPTION** This routine computes the quotient and remainder of *numer*/*denom*. This routine is similar to `div()`, except that the arguments and the elements of the returned structure are all of type `long`.

This routine is not reentrant. For a reentrant version, see `ldiv_r()`.

**INCLUDE FILES** `stdlib.h`

**RETURNS** A structure of type `ldiv_t`, containing both the quotient and the remainder.

**SEE ALSO** [ansiStdlib](#)

---

## ldiv\_r()

**NAME** `ldiv_r()` – compute a quotient and remainder (reentrant)

**SYNOPSIS**

```
void ldiv_r
(
    long    numer,          /* numerator */
    long    denom,         /* denominator */
    ldiv_t * divStructPtr  /* ldiv_t structure */
)
```

<b>DESCRIPTION</b>	This routine computes the quotient and remainder of <i>numer/denom</i> . The quotient and remainder are stored in the <b>ldiv_t</b> structure <b>divStructPtr</b> . This routine is the reentrant version of <b>ldiv()</b> .
<b>INCLUDE FILES</b>	<b>stdlib.h</b>
<b>RETURNS</b>	N/A
<b>SEE ALSO</b>	<b>ansiStdlib</b>

---

## ledClose()

<b>NAME</b>	<b>ledClose()</b> – discard the line-editor ID
<b>SYNOPSIS</b>	<pre>STATUS ledClose (     int led_id          /* ID returned by ledOpen */ )</pre>
<b>DESCRIPTION</b>	This routine frees resources allocated by <b>ledOpen()</b> . The low-level input/output file descriptors are not closed.
<b>RETURNS</b>	OK.
<b>SEE ALSO</b>	<b>ledLib, ledOpen()</b>

---

## ledControl()

<b>NAME</b>	<b>ledControl()</b> – change the line-editor ID parameters
<b>SYNOPSIS</b>	<pre>void ledControl (     int led_id,          /* ID returned by ledOpen */     int inFd,           /* new input fd (NONE = no change) */     int outFd,          /* new output fd (NONE = no change) */     int histSize        /* new history list size (NONE = no */                        /* change), (0 = display) */ )</pre>

<b>DESCRIPTION</b>	This routine changes the input/output file descriptor and the size of the history list.
<b>RETURNS</b>	N/A
<b>SEE ALSO</b>	<b>ledLib</b>

---

## ledOpen()

**NAME** ledOpen() – create a new line-editor ID

**SYNOPSIS**

```
int ledOpen
(
    int inFd,                /* low-level device input fd */
    int outFd,              /* low-level device output fd */
    int histSize            /* size of history list */
)
```

**DESCRIPTION** This routine creates the ID that is used by **ledRead()**, **ledClose()**, and **ledControl()**. Storage is allocated for up to *histSize* previously read lines.

**RETURNS** The line-editor ID, or **ERROR** if the routine runs out of memory.

**SEE ALSO** **ledLib**, **ledRead()**, **ledClose()**, **ledControl()**

---

## ledRead()

**NAME** ledRead() – read a line with line-editing

**SYNOPSIS**

```
int ledRead
(
    int led_id,              /* ID returned by ledOpen */
    char * string,          /* where to return line */
    int maxBytes            /* maximum number of chars to read */
)
```

**DESCRIPTION** This routine handles line-editing and history substitutions. If the low-level input file descriptor is not in **OPT\_LINE** mode, only an ordinary **read()** routine will be performed.

**RETURNS** The number of characters read, or EOF.

**SEE ALSO** `ledLib`

---

## lio\_listio()

**NAME** `lio_listio()` – initiate a list of asynchronous I/O requests (POSIX)

**SYNOPSIS**

```
int lio_listio
(
    int          mode,      /* LIO_WAIT or LIO_NOWAIT */
    struct aiocb * list[], /* list of operations */
    int          nEnt,     /* size of list */
    struct sigevent * pSig /* signal on completion */
)
```

**DESCRIPTION** This routine submits a number of I/O operations (up to `AIO_LISTIO_MAX`) to be performed asynchronously. *list* is a pointer to an array of `aiocb` structures that specify the AIO operations to be performed. The array is of size *nEnt*.

The `aio_lio_opcode` field of the `aiocb` structure specifies the AIO operation to be performed. Valid entries include `LIO_READ`, `LIO_WRITE`, and `LIO_NOP`. `LIO_READ` corresponds to a call to `aio_read()`, `LIO_WRITE` corresponds to a call to `aio_write()`, and `LIO_NOP` is ignored.

The *mode* argument can be either `LIO_WAIT` or `LIO_NOWAIT`. If *mode* is `LIO_WAIT`, `lio_listio()` does not return until all the AIO operations complete and the *pSig* argument is ignored. If *mode* is `LIO_NOWAIT`, the `lio_listio()` returns as soon as the operations are queued. In this case, if *pSig* is not `NULL` and the signal number indicated by `pSig>sigev_signo` is not zero, the signal `pSig>sigev_signo` is delivered when all requests have completed.

**RETURNS** `OK` if requests queued successfully, otherwise `ERROR`.

**ERRNO** `EINVAL`, `EAGAIN`, `EIO`

**INCLUDE FILES** `aio.h`

**SEE ALSO** `aioPxLib`, `aio_read()`, `aio_write()`, `aio_error()`, `aio_return()`

**listen()**

---

**listen()**

**NAME** `listen()` – enable connections to a socket

**SYNOPSIS**

```
STATUS listen
(
    int s,                /* socket descriptor */
    int backlog           /* number of connections to queue */
)
```

**DESCRIPTION** This routine enables connections to a socket. It also specifies the maximum number of unaccepted connections that can be pending at one time (*backlog*). After enabling connections with `listen()`, connections are actually accepted by `accept()`.

**RETURNS** OK, or ERROR if the socket is invalid or unable to listen.

**SEE ALSO** `sockLib`

---

**lkAddr()**

**NAME** `lkAddr()` – list symbols whose values are near a specified value

**SYNOPSIS**

```
void lkAddr
(
    unsigned int addr     /* address around which to look */
)
```

**DESCRIPTION** This command lists the symbols in the system symbol table that are near a specified value. The symbols that are displayed include:

- symbols whose values are immediately less than the specified value
- symbols with the specified value
- succeeding symbols, until at least 12 symbols have been displayed

This command also displays symbols that are local, *i.e.*, symbols found in the system symbol table only because their module was loaded by `ld()`.

**RETURNS** N/A

**SEE ALSO** `usrLib`, `symLib`, `symEach()`, *VxWorks Programmer's Guide: Target Shell*, `windsh`, *Tornado User's Guide: Shell*

---

## lkup()

**NAME** lkup() – list symbols

**SYNOPSIS**

```
void lkup
(
    char * substr          /* substring to match */
)
```

**DESCRIPTION** This command lists all symbols in the system symbol table whose names contain the string *substr*. If *substr* is omitted or is 0, a short summary of symbol table statistics is printed. If *substr* is the empty string (""), all symbols in the table are listed.

This command also displays symbols that are local, *i.e.*, symbols found in the system symbol table only because their module was loaded by **ld()**.

By default, **lkup()** displays 22 symbols at a time. This can be changed by modifying the global variable **symLkupPgSz**. If this variable is set to 0, **lkup()** displays all the symbols without interruption.

**RETURNS** N/A

**SEE ALSO** **usrLib**, **symLib**, **symEach()**, *VxWorks Programmer's Guide: Target Shell*, **windsh**, *Tornado User's Guide: Shell*

---

## ll()

**NAME** ll() – generate a long listing of directory contents

**SYNOPSIS**

```
STATUS ll
(
    char * dirName        /* name of directory to list */
)
```

**DESCRIPTION** This command causes a long listing of a directory's contents to be displayed. It is equivalent to:

```
-> dirList 1, dirName, TRUE, FALSE
```

*dirName* is a name of a directory or file, and may contain wildcards.

## **llr()**

---

**NOTE:** This is a target resident function, which manipulates the target I/O system. It must be preceded with the @ letter if executed from the Tornado Shell (**windsh**), which has a built-in command of the same name that operates on the Host's I/O system.

---

**NOTE:** When used with **netDrv** devices (FTP or RSH), **ll()** does not give directory information. It is equivalent to an **ls()** call with no long-listing option.

---

**RETURNS** OK or ERROR.

**SEE ALSO** **usrFsLib**, **dirList()**

---

## **llr()**

**NAME** **llr()** – do a long listing of directory and all its subdirectories contents

**SYNOPSIS**

```
STATUS llr  
(  
    char * dirName          /* name of directory to list */  
)
```

**DESCRIPTION** This command causes a long listing of a directory's contents to be displayed. It is equivalent to:

```
-> dirList 1, dirName, TRUE, TRUE
```

*dirName* is a name of a directory or file, and may contain wildcards.

---

**NOTE:** When used with **netDrv** devices (FTP or RSH), **ll()** does not give directory information. It is equivalent to an **ls()** call with no long-listing option.

---

**RETURNS** OK or ERROR.

**SEE ALSO** **usrFsLib**, **dirList()**

---

## loadModule()

<b>NAME</b>	<b>loadModule()</b> – load an object module into memory
<b>SYNOPSIS</b>	<pre>MODULE_ID loadModule (     int          fd,          /* fd of file to load */     int          symFlag     /* symbols to add to table (LOAD_[NO ALL]_SYMBOLS) */ GLOBAL ) </pre>
<b>DESCRIPTION</b>	<p>This routine loads an object module from the specified file, and places the code, data, and BSS into memory allocated from the system memory pool.</p> <p>This call is equivalent to <b>loadModuleAt()</b> with <b>NULL</b> for the addresses of text, data, and BSS segments. For more details, see the manual entry for <b>loadModuleAt()</b>.</p>
<b>RETURNS</b>	<b>MODULE_ID</b> , or <b>NULL</b> if the routine cannot read the file, there is not enough memory, or the file format is illegal.
<b>SEE ALSO</b>	<b>loadLib</b> , <b>loadModuleAt()</b>

---

## loadModuleAt()

<b>NAME</b>	<b>loadModuleAt()</b> – load an object module into memory
<b>SYNOPSIS</b>	<pre>MODULE_ID loadModuleAt (     int    fd,                /* fd from which to read module */     int    symFlag,          /* symbols to add to table (LOAD_[NO char * *ppText,             /* load text segment at addr. pointed to by */                                 /* this ptr, return load addr. via this ptr */     char * *ppData,         /* load data segment at addr. pointed to by */                                 /* this pointer, return load addr. via this */                                 /* ptr */     char * *ppBss           /* load BSS segment at addr. pointed to by */                                 /* this pointer, return load addr. via this */                                 /* ptr */ ) </pre>

**DESCRIPTION** This routine reads an object module from *fd*, and loads the code, data, and BSS segments at the specified load addresses in memory set aside by the user using **malloc()**, or in the system memory partition as described below. The module is properly relocated according to the relocation commands in the file. Unresolved externals will be linked to symbols found in the system symbol table. Symbols in the module being loaded can optionally be added to the system symbol table.

#### **LINKING UNRESOLVED EXTERNALS**

As the module is loaded, any unresolved external references are resolved by looking up the missing symbols in the system symbol table. If found, those references are correctly linked to the new module. If unresolved external references cannot be found in the system symbol table, then an error message (“undefined symbol: ...”) is printed for the symbol, but the loading/linking continues. The partially resolved module is not removed, to enable the user to examine the module for debugging purposes. Care should be taken when executing code from the resulting module. Executing code which contains references to unresolved symbols may have unexpected results and may corrupt the system’s memory.

Even though a module with unresolved symbols remains loaded after this routine returns, **NULL** will be returned to enable the caller to detect the failure programmatically. To unload the module, the caller may either call the unload routine with the module name, or look up the module using the module name and then unload the module using the returned **MODULE\_ID**. See the library entries for **moduleLib** and **unldLib** for details. The name of the module is the name of the file loaded with the path removed.

#### **ADDING SYMBOLS TO THE SYMBOL TABLE**

The symbols defined in the module to be loaded may be optionally added to the system symbol table, depending on the value of *symFlag*:

##### **LOAD\_NO\_SYMBOLS**

add no symbols to the system symbol table

##### **LOAD\_LOCAL\_SYMBOLS**

add only local symbols to the system symbol table

##### **LOAD\_GLOBAL\_SYMBOLS**

add only external symbols to the system symbol table

##### **LOAD\_ALL\_SYMBOLS**

add both local and external symbols to the system symbol table

##### **HIDDEN\_MODULE**

do not display the module via **moduleShow()**.

Obsolete symbols:

For backward compatibility with previous releases, the following symbols are also added to the symbol table to indicate the start of each segment: *filename\_text*, *filename\_data*, and *filename\_bss*, where *filename* is the name associated with the *fd*. Note that these symbols

are not available when the ELF format is used. Also they will disappear with the next VxWorks release. The **moduleLib** API should be used instead to get segment information.

**RELOCATION**

The relocation commands in the object module are used to relocate the text, data, and BSS segments of the module. The location of each segment can be specified explicitly, or left unspecified in which case memory will be allocated for the segment from the system memory partition. This is determined by the parameters *ppText*, *ppData*, and *ppBss*, each of which can have the following values:

**NULL**

no load address is specified, none will be returned;

A pointer to **LD\_NO\_ADDRESS**

no load address is specified, the return address is referenced by the pointer;

## A pointer to an address

the load address is specified.

The *ppText*, *ppData*, and *ppBss* parameters specify where to load the text, data, and bss sections respectively. Each of these parameters is a pointer to a pointer; for example, *\*\*ppText* gives the address where the text segment is to begin.

For any of the three parameters, there are two ways to request that new memory be allocated, rather than specifying the section's starting address: you can either specify the parameter itself as **NULL**, or you can write the constant **LD\_NO\_ADDRESS** in place of an address. In the second case, **loadModuleAt()** routine replaces the **LD\_NO\_ADDRESS** value with the address actually used for each section (that is, it records the address at *\*ppText*, *\*ppData*, or *\*ppBss*).

The double indirection not only permits reporting the addresses actually used, but also allows you to specify loading a segment at the beginning of memory, since the following cases can be distinguished:

- (1) Allocate memory for a section (text in this example): *ppText* == **NULL**
- (2) Begin a section at address zero (the text section, below): *\*ppText* == 0

Note that **loadModule()** is equivalent to this routine if all three of the segment-address parameters are set to **NULL**.

**COMMON**

Some host compiler/linker combinations use another storage class internally called "common". In the C language, uninitialized global variables are eventually put in the bss segment. However, in partially linked object modules they are flagged internally as "common" and the static linker (host) resolves these and places them in bss as a final step in creating a fully linked object module. However, the target loader is most often used to load partially linked object modules. When the target loader encounters a variable labeled "common", its behavior depends on the following flags:

**LOAD\_COMMON\_MATCH\_NONE**

Allocate memory for the variable with **malloc()** and enter the variable in the target symbol table (if specified) at that address. This is the default.

**LOAD\_COMMON\_MATCH\_USER**

Search for the symbol in the target symbol table, excluding the vxWorks image symbols. If several symbols exist, then the order of matching is: (1) bss, (2) data. If no symbol is found, act like the default.

**LOAD\_COMMON\_MATCH\_ALL**

Search for the symbol in the target symbol table, including the vxWorks image symbols. If several symbols exist, then the order of matching is: (1) bss, (2) data. If no symbol is found, act like the default.

Note that most UNIX loaders have an option that forces resolution of the common storage while leaving the module relocatable (for example, with typical BSD UNIX loaders, use options “-rd”).

**EXAMPLES**

Load a module into allocated memory, but do not return segment addresses:

```
module_id = loadModuleAt (fd, LOAD_GLOBAL_SYMBOLS, NULL, NULL, NULL);
```

Load a module into allocated memory, and return segment addresses:

```
pText = pData = pBss = LD_NO_ADDRESS;  
module_id = loadModuleAt (fd, LOAD_GLOBAL_SYMBOLS, &pText, &pData, &pBss);
```

Load a module to off-board memory at a specified address:

```
pText = 0x800000; /* address of text segment */  
pData = pBss = LD_NO_ADDRESS /* other segments follow by default */  
module_id = loadModuleAt (fd, LOAD_GLOBAL_SYMBOLS, &pText, &pData, &pBss);
```

**RETURNS**

MODULE\_ID, or NULL if the file cannot be read, there is not enough memory, the file format is illegal, or there were unresolved symbols.

**SEE ALSO**

**loadLib**, *VxWorks Programmer's Guide: Basic OS*

---

## localeconv()

**NAME**

**localeconv()** – set the components of an object with type **lconv** (ANSI)

**SYNOPSIS**

```
struct lconv *localeconv (void)
```

**DESCRIPTION**

This routine sets the components of an object with type **struct lconv** with values appropriate for the formatting of numeric quantities (monetary and otherwise) according to the rules of the current locale.

The members of the structure with type **char \*** are pointers to strings any of which (except **decimal\_point**) can point to "" to indicate that the value is not available in the current locale or is of zero length. The members with type **char** are nonnegative numbers, any of which can be **CHAR\_MAX** to indicate that the value is not available in the current locale. The members include the following:

**char \*decimal\_point**

The decimal-point character used to format non-monetary quantities.

**char \*thousands\_sep**

The character used to separate groups of digits before the decimal-point character in formatted non-monetary quantities.

**char \*grouping**

A string whose elements indicate the size of each group of digits in formatted non-monetary quantities.

**char \*int\_curr\_symbol**

The international currency symbol applicable to the current locale. The first three characters contain the alphabetic international currency symbol in accordance with those specified in ISO 4217:1987. The fourth character (immediately preceding the null character) is the character used to separate the international currency symbol from the monetary quantity.

**char \*currency\_symbol**

The local currency symbol applicable to the current locale.

**char \*mon\_decimal\_point**

The decimal-point used to format monetary quantities.

**char \*mon\_thousands\_sep**

The separator for groups of digits before the decimal-point in formatted monetary quantities.

**char \*mon\_grouping**

A string whose elements indicate the size of each group of digits in formatted monetary quantities.

**char \*positive\_sign**

The string used to indicate a nonnegative-valued formatted monetary quantity.

**char \*negative\_sign**

The string used to indicate a negative-valued formatted monetary quantity.

**char int\_frac\_digits**

The number of fractional digits (those after the decimal-point) to be displayed in an internationally formatted monetary quantity.

**char frac\_digits**

The number of fractional digits (those after the decimal-point) to be displayed in a formatted monetary quantity.

**localeconv( )**

char p\_cs\_precedes

Set to 1 or 0 if the **currency\_symbol** respectively precedes or succeeds the value for a nonnegative formatted monetary quantity.

char p\_sep\_by\_space

Set to 1 or 0 if the **currency\_symbol** respectively is or is not separated by a space from the value for a nonnegative formatted monetary quantity.

char n\_cs\_precedes

Set to 1 or 0 if the **currency\_symbol** respectively precedes or succeeds the value for a negative formatted monetary quantity.

char n\_sep\_by\_space

Set to 1 or 0 if the **currency\_symbol** respectively is or is not separated by a space from the value for a negative formatted monetary quantity.

char p\_sign\_posn

Set to a value indicating the positioning of the **positive\_sign** for a nonnegative formatted monetary quantity.

char n\_sign\_posn

Set to a value indicating the positioning of the **negative\_sign** for a negative formatted monetary quantity.

The elements of **grouping** and **mon\_grouping** are interpreted according to the following:

CHAR\_MAX

No further grouping is to be performed.

0

The previous element is to be repeatedly used for the remainder of the digits.

other

The integer value is the number of the digits that comprise the current group. The next element is examined to determine the size of the next group of digits before the current group.

The values of **p\_sign\_posn** and **n\_sign\_posn** are interpreted according to the following:

0

Parentheses surround the quantity and **currency\_symbol**.

1

The sign string precedes the quantity and **currency\_symbol**.

2

The sign string succeeds the quantity and **currency\_symbol**.

3

The sign string immediately precedes the **currency\_symbol**.

4

The sign string immediately succeeds the **currency\_symbol**.

The implementation behaves as if no library function calls **localeconv()**.

The **localeconv()** routine returns a pointer to the filled-in object. The structure pointed to by the return value is not modified by the program, but may be overwritten by a subsequent call to **localeconv()**. In addition, calls to **setlocale()** with categories **LC\_ALL**, **LC\_MONETARY**, or **LC\_NUMERIC** may overwrite the contents of the structure.

**INCLUDE FILES**    **locale.h, limits.h**

**RETURNS**        A pointer to the structure **lconv**.

**SEE ALSO**        **ansiLocale**

---

## localtime()

**NAME**            **localtime()** – convert calendar time into broken-down time (ANSI)

**SYNOPSIS**

```
struct tm *localtime
(
    const time_t * timer      /* calendar time in seconds */
)
```

**DESCRIPTION**    This routine converts the calendar time pointed to by *timer* into broken-down time, expressed as local time.

This routine is not reentrant. For a reentrant version, see **localtime\_r()**.

**INCLUDE FILES**    **time.h**

**RETURNS**        A pointer to a **tm** structure containing the local broken-down time.

**SEE ALSO**        **ansiTime**

## localtime\_r()

<b>NAME</b>	<b>localtime_r()</b> – convert calendar time into broken-down time (POSIX)
<b>SYNOPSIS</b>	<pre>int localtime_r (     const time_t * timer,      /* calendar time in seconds */     struct tm *   timeBuffer /* buffer for the broken-down time */ )</pre>
<b>DESCRIPTION</b>	<p>This routine converts the calendar time pointed to by <i>timer</i> into broken-down time, expressed as local time. The broken-down time is stored in <i>timeBuffer</i>.</p> <p>This routine is the POSIX re-entrant version of <b>localtime()</b>.</p>
<b>INCLUDE FILES</b>	<b>time.h</b>
<b>RETURNS</b>	OK.
<b>SEE ALSO</b>	<b>ansiTime</b>

---

## log()

<b>NAME</b>	<b>log()</b> – compute a natural logarithm (ANSI)
<b>SYNOPSIS</b>	<pre>double log (     double x                /* value to compute the natural logarithm of */ )</pre>
<b>DESCRIPTION</b>	<p>This routine returns the natural logarithm of <i>x</i> in double precision (IEEE double, 53 bits).</p> <p>A domain error occurs if the argument is negative. A range error may occur if the argument is zero.</p>
<b>INCLUDE FILES</b>	<b>math.h</b>
<b>RETURNS</b>	<p>The double-precision natural logarithm of <i>x</i>.</p> <p>Special cases:</p> <ul style="list-style-type: none"><li>If <i>x</i> &lt; 0 (including -INF), it returns NaN with signal.</li><li>If <i>x</i> is +INF, it returns <i>x</i> with no signal.</li></ul>

If  $x$  is 0, it returns -INF with signal.  
If  $x$  is NaN it returns  $x$  with no signal.

**SEE ALSO**      **ansiMath, mathALib**

---

## log2()

**NAME**            **log2()** – compute a base-2 logarithm

**SYNOPSIS**        **double log2**  
                  (  
                  **double x** /\* value to compute the base-two logarithm of \*/  
                  )

**DESCRIPTION**    This routine returns the base-2 logarithm of  $x$  in double precision.

**INCLUDE FILES**   **math.h**

**RETURNS**        The double-precision base-2 logarithm of  $x$ .

**SEE ALSO**        **mathALib**

---

## log2f()

**NAME**            **log2f()** – compute a base-2 logarithm

**SYNOPSIS**        **float log2f**  
                  (  
                  **float x**    /\* value to compute the base-2 logarithm of \*/  
                  )

**DESCRIPTION**    This routine returns the base-2 logarithm of  $x$  in single precision.

**INCLUDE FILES**   **math.h**

**RETURNS**        The single-precision base-2 logarithm of  $x$ .

**SEE ALSO**        **mathALib**

## log10( )

<b>NAME</b>	<b>log10( )</b> – compute a base-10 logarithm (ANSI)
<b>SYNOPSIS</b>	<pre>double log10 (     double x          /* value to compute the base-10 logarithm of */ )</pre>
<b>DESCRIPTION</b>	This routine returns the base 10 logarithm of $x$ in double precision (IEEE double, 53 bits). A domain error occurs if the argument is negative. A range error may if the argument is zero.
<b>INCLUDE FILES</b>	<b>math.h</b>
<b>RETURNS</b>	The double-precision base-10 logarithm of $x$ . Special cases: If $x < 0$ , <b>log10( )</b> returns NaN with signal. if $x$ is +INF, it returns $x$ with no signal. if $x$ is 0, it returns -INF with signal. if $x$ is NaN it returns $x$ with no signal.
<b>SEE ALSO</b>	<b>ansiMath, mathALib</b>

---

## log10f( )

<b>NAME</b>	<b>log10f( )</b> – compute a base-10 logarithm (ANSI)
<b>SYNOPSIS</b>	<pre>float log10f (     float x          /* value to compute the base-10 logarithm of */ )</pre>
<b>DESCRIPTION</b>	This routine returns the base-10 logarithm of $x$ in single precision.
<b>INCLUDE FILES</b>	<b>math.h</b>
<b>RETURNS</b>	The single-precision base-10 logarithm of $x$ .
<b>SEE ALSO</b>	<b>mathALib</b>

---

## logf()

**NAME** `logf()` – compute a natural logarithm (ANSI)

**SYNOPSIS**

```
float logf
(
    float x /* value to compute the natural logarithm of */
)
```

**DESCRIPTION** This routine returns the logarithm of *x* in single precision.

**INCLUDE FILES** `math.h`

**RETURNS** The single-precision natural logarithm of *x*.

**SEE ALSO** `mathALib`

---

## logFdAdd()

**NAME** `logFdAdd()` – add a logging file descriptor

**SYNOPSIS**

```
STATUS logFdAdd
(
    int fd /* file descriptor for additional logging */
          /* device */
)
```

**DESCRIPTION** This routine adds to the log file descriptor list another file descriptor *fd* to which messages will be logged. The file descriptor must be a valid open file descriptor.

**RETURNS** `OK`, or `ERROR` if the allowable number of additional logging file descriptors (5) is exceeded.

**SEE ALSO** `logLib`, `logFdDelete()`

## logFdDelete()

<b>NAME</b>	logFdDelete() – delete a logging file descriptor
<b>SYNOPSIS</b>	<pre>STATUS logFdDelete (     int fd                /* file descriptor to stop using as logging */                         /* device */ )</pre>
<b>DESCRIPTION</b>	This routine removes from the log file descriptor list a logging file descriptor added by <b>logFdAdd()</b> . The file descriptor is not closed; but is no longer used by the logging facilities.
<b>RETURNS</b>	OK, or ERROR if the file descriptor was not added with <b>logFdAdd()</b> .
<b>SEE ALSO</b>	logLib, logFdAdd()

---

## logFdSet()

<b>NAME</b>	logFdSet() – set the primary logging file descriptor
<b>SYNOPSIS</b>	<pre>void logFdSet (     int fd                /* file descriptor to use as logging device */ )</pre>
<b>DESCRIPTION</b>	<p>This routine changes the file descriptor where messages from <b>logMsg()</b> are written, allowing the log device to be changed from the default specified by <b>logInit()</b>. It first removes the old file descriptor (if one had been previously set) from the log file descriptor list, then adds the new <i>fd</i>.</p> <p>The old logging file descriptor is not closed or affected by this call; it is simply no longer used by the logging facilities.</p>
<b>RETURNS</b>	N/A
<b>SEE ALSO</b>	logLib, logFdAdd(), logFdDelete()

---

## loginDefaultEncrypt()

**NAME** loginDefaultEncrypt() – default password encryption routine

**SYNOPSIS**

```
STATUS loginDefaultEncrypt
(
    char * in,           /* input string */
    char * out          /* encrypted string */
)
```

**DESCRIPTION** This routine provides default encryption for login passwords. It employs a simple encryption algorithm. It takes as arguments a string *in* and a pointer to a buffer *out*. The encrypted string is then stored in the buffer.

The input strings must be at least 8 characters and no more than 40 characters.

If a more sophisticated encryption algorithm is needed, this routine can be replaced, as long as the new encryption routine retains the same declarations as the default routine. The utility **vxencrypt** in *host/hostOs/bin* should also be replaced by a host version of *encryptRoutine*. For more information, see the manual entry for **loginEncryptInstall()**.

**RETURNS** OK, or ERROR if the password is invalid.

**SEE ALSO** loginLib, loginEncryptInstall(), vxencrypt

---

## loginEncryptInstall()

**NAME** loginEncryptInstall() – install an encryption routine

**SYNOPSIS**

```
void loginEncryptInstall
(
    FUNCPTR rtn,        /* function pointer to encryption routine */
    int    var         /* argument to the encryption routine (unused) */
)
```

**DESCRIPTION** This routine allows the user to install a custom encryption routine. The custom routine *rtn* must be of the following form:

```
STATUS encryptRoutine
(
    char *password,    /* string to encrypt */
```

```
        char *encryptedPassword      /* resulting encryption */  
    )
```

When a custom encryption routine is installed, a host version of this routine must be written to replace the tool `vxencrypt()` in `host/hostOs/bin`.

**EXAMPLE** The custom example above could be installed as follows:

```
#ifdef INCLUDE_SECURITY  
    loginInit ();                /* initialize login table */  
    shellLoginInstall (loginPrompt, NULL); /* install shell security */  
    loginEncryptInstall (encryptRoutine, NULL); /* install encrypt routine */  
#endif
```

**RETURNS** N/A

**SEE ALSO** `loginLib`, `loginDefaultEncrypt()`, `vxencrypt`

---

## loginInit()

**NAME** `loginInit()` – initialize the login table

**SYNOPSIS** `void loginInit (void)`

**DESCRIPTION** This routine must be called to initialize the login data structure used by routines throughout this module. If the configuration macro `INCLUDE_SECURITY` is defined, it is called by `usrRoot()` in `usrConfig.c`, before any other routines in this module.

**RETURNS** N/A

**SEE ALSO** `loginLib`

---

## logInit()

<b>NAME</b>	<b>logInit()</b> – initialize message logging library
<b>SYNOPSIS</b>	<pre><b>STATUS</b> logInit (     <b>int</b> fd,                /* file descriptor to use as logging device */     <b>int</b> maxMsgs           /* max. number of messages allowed in log queue */ )</pre>
<b>DESCRIPTION</b>	<p>This routine specifies the file descriptor to be used as the logging device and the number of messages that can be in the logging queue. If more than <i>maxMsgs</i> are in the queue, they will be discarded. A message is printed to indicate lost messages.</p> <p>This routine spawns <b>logTask()</b>, the task-level portion of error logging.</p> <p>This routine must be called before any other routine in <b>logLib</b>. This is done by the root task, <b>usrRoot()</b>, in <b>usrConfig.c</b>.</p>
<b>RETURNS</b>	OK, or <b>ERROR</b> if a message queue could not be created or <b>logTask()</b> could not be spawned.
<b>SEE ALSO</b>	<b>logLib</b>

---

## loginPrompt()

<b>NAME</b>	<b>loginPrompt()</b> – display a login prompt and validate a user entry
<b>SYNOPSIS</b>	<pre><b>STATUS</b> loginPrompt (     <b>char</b> * userName       /* user name, ask if NULL or not provided */ )</pre>
<b>DESCRIPTION</b>	<p>This routine displays a login prompt and validates a user entry. If both user name and password match with an entry in the login table, the user is then given access to the VxWorks system. Otherwise, it prompts the user again.</p> <p>All control characters are disabled during authentication except CTRL-D, which will terminate the remote login session.</p>

- RETURNS** OK if the name and password are valid, or **ERROR** if there is an EOF or the routine times out.
- SEE ALSO** loginLib

---

## loginStringSet()

- NAME** loginStringSet() – change the login string
- SYNOPSIS**
- ```
void loginStringSet
(
    char * newString          /* string to become new login prompt */
)
```
- DESCRIPTION** This routine changes the login prompt string to *newString*. The maximum string length is 80 characters.
- RETURNS** N/A
- SEE ALSO** loginLib

---

## loginUserAdd()

- NAME** loginUserAdd() – add a user to the login table
- SYNOPSIS**
- ```
STATUS loginUserAdd
(
    char name[MAX_LOGIN_NAME_LEN+1], /* user name */
    char passwd[80]                  /* user password */
)
```
- DESCRIPTION** This routine adds a user name and password entry to the login table. Note that what is saved in the login table is the user name and the address of *passwd*, not the actual password.
- The length of user names should not exceed `MAX_LOGIN_NAME_LEN`, while the length of passwords depends on the encryption routine used. For the default encryption routine, passwords should be at least 8 characters long and no more than 40 characters.

The procedure for adding a new user to login table is as follows:

- (1) Generate the encrypted password by invoking **vxencrypt** in **host/hostOs/bin**.
- (2) Add a user by invoking **loginUserAdd()** in the VxWorks shell with the user name and the encrypted password.

The password of a user can be changed by first deleting the user entry, then adding the user entry again with the new encrypted password.

**EXAMPLE**

```
-> loginUserAdd "peter", "RRdRd9Qbyz"
value = 0 = 0x0
-> loginUserAdd "robin", "bSzyyqbsb"
value = 0 = 0x0
-> loginUserShow
    User Name
    =====
    peter
    robin
value = 0 = 0x0
->
```

**RETURNS**

OK, or **ERROR** if the user name has already been entered.

**SEE ALSO**

**loginLib**, **vxencrypt**

---

## loginUserDelete()

**NAME**

**loginUserDelete()** – delete a user entry from the login table

**SYNOPSIS**

```
STATUS loginUserDelete
(
    char * name,           /* user name */
    char * passwd         /* user password */
)
```

**DESCRIPTION**

This routine deletes an entry in the login table. Both the user name and password must be specified to remove an entry from the login table.

**RETURNS**

OK, or **ERROR** if the specified user or password is incorrect.

**SEE ALSO**

**loginLib**

## loginUserShow()

**NAME** loginUserShow() – display the user login table

**SYNOPSIS** void loginUserShow (void)

**DESCRIPTION** This routine displays valid user names.

**EXAMPLE**

```
-> loginUserShow ()
    User Name
    =====
    peter
    robin
    value = 0 = 0x0
```

**RETURNS** N/A

**SEE ALSO** loginLib

---

## loginUserVerify()

**NAME** loginUserVerify() – verify a user name and password in the login table

**SYNOPSIS**

```
STATUS loginUserVerify
(
    char * name,           /* name of user */
    char * passwd         /* password of user */
)
```

**DESCRIPTION** This routine verifies a user entry in the login table.

**RETURNS** OK, or ERROR if the user name or password is not found.

**SEE ALSO** loginLib

---

## logMsg()

**NAME** logMsg() – log a formatted error message

**SYNOPSIS**

```
int logMsg
(
    char * fmt,          /* format string for print */
    int   arg1,         /* first of six required args for fmt */
    int   arg2,
    int   arg3,
    int   arg4,
    int   arg5,
    int   arg6
)
```

**DESCRIPTION** This routine logs a specified message via the logging task. This routine's syntax is similar to **printf()** -- a format string is followed by arguments to format. However, **logMsg()** takes a **char \*** rather than a **const char \*** and requires a fixed number of arguments (6).

The task ID of the caller is prepended to the specified message.

### SPECIAL CONSIDERATIONS

Because **logMsg()** does not actually perform the output directly to the logging streams, but instead queues the message to the logging task, **logMsg()** can be called from interrupt service routines.

However, since the arguments are interpreted by the **logTask()** at the time of actual logging, instead of at the moment when **logMsg()** is called, arguments to **logMsg()** should not be pointers to volatile entities (*e.g.*, dynamic strings on the caller stack).

**logMsg()** checks to see whether or not it is running in interrupt context. If it is, it will not block. However, if invoked from a task, it can cause the task to block.

For more detailed information about the use of **logMsg()**, see the manual entry for **logLib**.

**EXAMPLE** If the following code were executed by task 20:

```
{
    name = "GRONK";
    num = 123;
    logMsg ("ERROR - name = %s, num = %d.\n", name, num, 0, 0, 0, 0);
}
```

the following error message would appear on the system log:

```
0x180400 (t20): ERROR - name = GRONK, num = 123.
```

- RETURNS** The number of bytes written to the log queue, or EOF if the routine is unable to write a message.
- SEE ALSO** `logLib`, `printf()`, `logTask()`

---

## logout()

- NAME** `logout()` – log out of the VxWorks system
- SYNOPSIS** `void logout (void)`
- DESCRIPTION** This command logs out of the VxWorks shell. If a remote login is active (via `rlogin` or `telnet`), it is stopped, and standard I/O is restored to the console.
- SEE ALSO** `usrLib`, `rlogin()`, `telnet()`, `shellLogout()`, *VxWorks Programmer's Guide: Target Shell*

---

## logTask()

- NAME** `logTask()` – message-logging support task
- SYNOPSIS** `void logTask (void)`
- DESCRIPTION** This routine prints the messages logged with `logMsg()`. It waits on a message queue and prints the messages as they arrive on the file descriptor specified by `logInit()` (or a subsequent call to `logFdSet()` or `logFdAdd()`).
- This task is spawned by `logInit()`.
- RETURNS** N/A
- SEE ALSO** `logLib`, `logMsg()`

---

## longjmp()

**NAME** `longjmp()` – perform non-local goto by restoring saved environment (ANSI)

**SYNOPSIS**

```
void longjmp
(
    jmp_buf env,
    int val
)
```

**DESCRIPTION** This routine restores the environment saved by the most recent invocation of `setjmp()` that used the same `jmp_buf` specified in the argument `env`. The restored environment includes the program counter, thus transferring control to the `setjmp()` caller.

If there was no corresponding `setjmp()` call, or if the call containing the corresponding `setjmp()` has already returned, the behavior of `longjmp()` is unpredictable.

All accessible objects in memory retain their values as of the time `longjmp()` was called, with one exception: local objects on the C stack that are not declared `volatile`, and have been changed between the `setjmp()` invocation and the `longjmp()` call, have unpredictable values.

The `longjmp()` function executes correctly in contexts of signal handlers and any of their associated functions (but not from interrupt handlers).

---

**WARNING:** Do not use `longjmp()` or `setjmp()` from an ISR.

---

**RETURNS** This routine does not return to its caller. Instead, it causes `setjmp()` to return `val`, unless `val` is 0; in that case `setjmp()` returns 1.

**SEE ALSO** `ansiSetjmp`, `setjmp()`

---

## ls()

**NAME** `ls()` – generate a brief listing of a directory

**SYNOPSIS**

```
STATUS ls
(
    char * dirName,          /* name of dir to list */
    BOOL doLong              /* switch on details */
)
```

***lseek()***

<b>DESCRIPTION</b>	<p>This function is simply a front-end for <b>dirList()</b>, intended for brevity and backward compatibility. It produces a list of files and directories, without details such as file size and date, and without recursion into subdirectories.</p> <p><i>dirName</i> is a name of a directory or file, and may contain wildcards. <i>doLong</i> is provided for backward compatibility.</p> <hr/> <p><b>NOTE:</b> This is a target resident function, which manipulates the target I/O system. It must be preceded with the @ letter if executed from the Tornado Shell (<b>windsh</b>), which has a built-in command of the same name that operates on the Host's I/O system.</p> <hr/>
<b>RETURNS</b>	OK or ERROR.
<b>SEE ALSO</b>	usrFsLib, dirList()

---

## ***lseek()***

<b>NAME</b>	<b><i>lseek()</i></b> – set a file read/write pointer
<b>SYNOPSIS</b>	<pre>int lseek (     int fd,                /* file descriptor */     long offset,           /* new byte offset to seek to */     int whence             /* relative file position */ )</pre>
<b>DESCRIPTION</b>	<p>This routine sets the file read/write pointer of file <i>fd</i> to <i>offset</i>. The argument <i>whence</i>, which affects the file position pointer, has three values:</p> <p><b>SEEK_SET</b> (0) - set to <i>offset</i>  <b>SEEK_CUR</b> (1) - set to current position plus <i>offset</i>  <b>SEEK_END</b> (2) - set to the size of the file plus <i>offset</i></p> <p>This routine calls <b>ioctl()</b> with functions <b>FIOWHERE</b>, <b>FIONREAD</b>, and <b>FIOSEEK</b>.</p>
<b>RETURNS</b>	The new offset from the beginning of the file, or <b>ERROR</b> .
<b>SEE ALSO</b>	<b>ioLib</b>

---

## lsr()

**NAME** `lsr()` – list the contents of a directory and any of its subdirectories

**SYNOPSIS**

```
STATUS lsr  
(  
    char * dirName           /* name of dir to list */  
)
```

**DESCRIPTION** This function is simply a front-end for `dirList()`, intended for brevity and backward compatibility. It produces a list of files and directories, without details such as file size and date, with recursion into subdirectories.

*dirName* is a name of a directory or file, and may contain wildcards.

**RETURNS** OK or ERROR.

**SEE ALSO** `usrFsLib`, `dirList()`

---

## lstAdd()

**NAME** `lstAdd()` – add a node to the end of a list

**SYNOPSIS**

```
void lstAdd  
(  
    LIST * pList,           /* pointer to list descriptor */  
    NODE * pNode         /* pointer to node to be added */  
)
```

**DESCRIPTION** This routine adds a specified node to the end of a specified list.

**RETURNS** N/A

**SEE ALSO** `lstLib`

## **lstConcat()**

<b>NAME</b>	<b>lstConcat()</b> – concatenate two lists
<b>SYNOPSIS</b>	<pre>void lstConcat (     LIST * pDstList,          /* destination list */     LIST * pAddList          /* list to be added to dstList */ )</pre>
<b>DESCRIPTION</b>	This routine concatenates the second list to the end of the first list. The second list is left empty. Either list (or both) can be empty at the beginning of the operation.
<b>RETURNS</b>	N/A
<b>SEE ALSO</b>	<b>lstLib</b>

---

## **lstCount()**

<b>NAME</b>	<b>lstCount()</b> – report the number of nodes in a list
<b>SYNOPSIS</b>	<pre>int lstCount (     LIST * pList              /* pointer to list descriptor */ )</pre>
<b>DESCRIPTION</b>	This routine returns the number of nodes in a specified list.
<b>RETURNS</b>	The number of nodes in the list.
<b>SEE ALSO</b>	<b>lstLib</b>

---

## lstDelete()

**NAME** `lstDelete()` – delete a specified node from a list

**SYNOPSIS**

```
void lstDelete
(
    LIST * pList,           /* pointer to list descriptor */
    NODE * pNode           /* pointer to node to be deleted */
)
```

**DESCRIPTION** This routine deletes a specified node from a specified list.

**RETURNS** N/A

**SEE ALSO** `lstLib`

---

## lstExtract()

**NAME** `lstExtract()` – extract a sublist from a list

**SYNOPSIS**

```
void lstExtract
(
    LIST * pSrcList,       /* pointer to source list */
    NODE * pStartNode,    /* first node in sublist to be extracted */
    NODE * pEndNode,      /* last node in sublist to be extracted */
    LIST * pDstList        /* ptr to list where to put extracted list */
)
```

**DESCRIPTION** This routine extracts the sublist that starts with *pStartNode* and ends with *pEndNode* from a source list. It places the extracted list in *pDstList*.

**RETURNS** N/A

**SEE ALSO** `lstLib`

## lstFind()

<b>NAME</b>	lstFind() – find a node in a list
<b>SYNOPSIS</b>	<pre>int lstFind (     LIST * pList,          /* list in which to search */     NODE * pNode          /* pointer to node to search for */ )</pre>
<b>DESCRIPTION</b>	This routine returns the node number of a specified node (the first node is 1).
<b>RETURNS</b>	The node number, or <b>ERROR</b> if the node is not found.
<b>SEE ALSO</b>	lstLib

---

## lstFirst()

<b>NAME</b>	lstFirst() – find first node in list
<b>SYNOPSIS</b>	<pre>NODE *lstFirst (     LIST * pList          /* pointer to list descriptor */ )</pre>
<b>DESCRIPTION</b>	This routine finds the first node in a linked list.
<b>RETURNS</b>	A pointer to the first node in a list, or <b>NULL</b> if the list is empty.
<b>SEE ALSO</b>	lstLib

---

## lstFree()

<b>NAME</b>	lstFree() – free up a list
<b>SYNOPSIS</b>	<pre>void lstFree (     LIST * pList          /* list for which to free all nodes */ )</pre>
<b>DESCRIPTION</b>	This routine turns any list into an empty list. It also frees up memory used for nodes.
<b>RETURNS</b>	N/A
<b>SEE ALSO</b>	lstLib, free()

---

## lstGet()

<b>NAME</b>	lstGet() – delete and return the first node from a list
<b>SYNOPSIS</b>	<pre>NODE *lstGet (     LIST * pList          /* ptr to list from which to get node */ )</pre>
<b>DESCRIPTION</b>	This routine gets the first node from a specified list, deletes the node from the list, and returns a pointer to the node gotten.
<b>RETURNS</b>	A pointer to the node gotten, or NULL if the list is empty.
<b>SEE ALSO</b>	lstLib

## lstInit()

**NAME**            **lstInit()** – initialize a list descriptor

**SYNOPSIS**        **void lstInit**  
                  (  
                  **LIST \* pList**                    */\* ptr to list descriptor to be initialized \*/*  
                  )

**DESCRIPTION**    This routine initializes a specified list to an empty list.

**RETURNS**        N/A

**SEE ALSO**        **lstLib**

---

## lstInsert()

**NAME**            **lstInsert()** – insert a node in a list after a specified node

**SYNOPSIS**        **void lstInsert**  
                  (  
                  **LIST \* pList,**                    */\* pointer to list descriptor \*/*  
                  **NODE \* pPrev,**                    */\* pointer to node after which to insert \*/*  
                  **NODE \* pNode**                    */\* pointer to node to be inserted \*/*  
                  )

**DESCRIPTION**    This routine inserts a specified node in a specified list. The new node is placed following the list node *pPrev*. If *pPrev* is **NULL**, the node is inserted at the head of the list.

**RETURNS**        N/A

**SEE ALSO**        **lstLib**

---

## lstLast()

**NAME** `lstLast()` – find the last node in a list

**SYNOPSIS**

```
NODE *lstLast  
(  
    LIST * pList          /* pointer to list descriptor */  
)
```

**DESCRIPTION** This routine finds the last node in a list.

**RETURNS** A pointer to the last node in the list, or `NULL` if the list is empty.

**SEE ALSO** `lstLib`

---

## lstLibInit()

**NAME** `lstLibInit()` – initializes `lstLib` module

**SYNOPSIS**

```
void lstLibInit (void)
```

**DESCRIPTION** This routine pulls `lstLib` into the vxWorks image.

**RETURNS** N/A

**SEE ALSO** `lstLib`

## lstNext()

<b>NAME</b>	lstNext() – find the next node in a list
<b>SYNOPSIS</b>	<pre>NODE *lstNext (     NODE * pNode           /* ptr to node whose successor is to be found */ ) </pre>
<b>DESCRIPTION</b>	This routine locates the node immediately following a specified node.
<b>RETURNS</b>	A pointer to the next node in the list, or NULL if there is no next node.
<b>SEE ALSO</b>	lstLib

---

## lstNStep()

<b>NAME</b>	lstNStep() – find a list node <i>nStep</i> steps away from a specified node
<b>SYNOPSIS</b>	<pre>NODE *lstNStep (     NODE * pNode,         /* the known node */     int   nStep           /* number of steps away to find */ ) </pre>
<b>DESCRIPTION</b>	This routine locates the node <i>nStep</i> steps away in either direction from a specified node. If <i>nStep</i> is positive, it steps toward the tail. If <i>nStep</i> is negative, it steps toward the head. If the number of steps is out of range, NULL is returned.
<b>RETURNS</b>	A pointer to the node <i>nStep</i> steps away, or NULL if the node is out of range.
<b>SEE ALSO</b>	lstLib

---

## lstNth()

**NAME** `lstNth()` – find the Nth node in a list

**SYNOPSIS**

```
NODE *lstNth
(
    LIST * pList,          /* pointer to list descriptor */
    int  nodenum          /* number of node to be found */
)
```

**DESCRIPTION** This routine returns a pointer to the node specified by a number *nodenum* where the first node in the list is numbered 1. Note that the search is optimized by searching forward from the beginning if the node is closer to the head, and searching back from the end if it is closer to the tail.

**RETURNS** A pointer to the Nth node, or NULL if there is no Nth node.

**SEE ALSO** `lstLib`

---

## lstPrevious()

**NAME** `lstPrevious()` – find the previous node in a list

**SYNOPSIS**

```
NODE *lstPrevious
(
    NODE * pNode          /* ptr to node whose predecessor is to be found */
)
```

**DESCRIPTION** This routine locates the node immediately preceding the node pointed to by *pNode*.

**RETURNS** A pointer to the previous node in the list, or NULL if there is no previous node.

**SEE ALSO** `lstLib`

**m()**


---

<b>NAME</b>	<b>m()</b> – modify memory
<b>SYNOPSIS</b>	<pre>void m (     void * adrs,          /* address to change */     int   width          /* width of unit to be modified (1, 2, 4, 8) */ )</pre>
<b>DESCRIPTION</b>	<p>This command prompts the user for modifications to memory in byte, short word, or long word specified by <i>width</i>, starting at the specified address. It prints each address and the current contents of that address, in turn. If <i>adrs</i> or <i>width</i> is zero or absent, it defaults to the previous value. The user can respond in one of several ways:</p> <p><b>RETURN</b></p> <p>Do not change this address, but continue, prompting at the next address.</p> <p>number Set the content of this address to <i>number</i>.</p> <p>. (dot) Do not change this address, and quit.</p> <p>EOF Do not change this address, and quit.</p> <p>All numbers entered and displayed are in hexadecimal.</p>
<b>RETURNS</b>	N/A
<b>SEE ALSO</b>	<b>usrLib</b> , <b>mRegs()</b> , <i>VxWorks Programmer's Guide: Target Shell</i> , <b>windsh</b> , <i>Tornado User's Guide: Shell</i>

---

**m2Delete()**

<b>NAME</b>	<b>m2Delete()</b> – delete all the MIB-II library groups
<b>SYNOPSIS</b>	<b>STATUS</b> <b>m2Delete (void)</b>
<b>DESCRIPTION</b>	This routine cleans up the state associated with the MIB-II library.
<b>RETURNS</b>	OK (always).
<b>SEE ALSO</b>	<b>m2Lib</b> , <b>m2SysDelete()</b> , <b>m2TcpDelete()</b> , <b>m2UdpDelete()</b> , <b>m2IcmpDelete()</b> , <b>m2IfDelete()</b> , <b>m2IpDelete()</b>

---

## m2IcmpDelete()

<b>NAME</b>	<b>m2IcmpDelete()</b> – delete all resources used to access the ICMP group
<b>SYNOPSIS</b>	<b>STATUS</b> m2IcmpDelete (void)
<b>DESCRIPTION</b>	This routine frees all the resources allocated at the time the ICMP group was initialized. The ICMP group should not be accessed after this routine has been called.
<b>RETURNS</b>	OK, always.
<b>SEE ALSO</b>	m2IcmpLib, m2IcmpInit(), m2IcmpGroupInfoGet()

---

## m2IcmpGroupInfoGet()

<b>NAME</b>	<b>m2IcmpGroupInfoGet()</b> – get the MIB-II ICMP-group global variables
<b>SYNOPSIS</b>	<b>STATUS</b> m2IcmpGroupInfoGet ( M2_ICMP * pIcmpInfo       /* pointer to the ICMP group structure */ )
<b>DESCRIPTION</b>	This routine fills in the ICMP structure at <i>pIcmpInfo</i> with the MIB-II ICMP scalar variables.
<b>RETURNS</b>	OK, or <b>ERROR</b> if the input parameter <i>pIcmpInfo</i> is invalid.
<b>ERRNO</b>	S_m2Lib_INVALID_PARAMETER
<b>SEE ALSO</b>	m2IcmpLib, m2IcmpInit(), m2IcmpDelete()

**M**

---

## m2IcmpInit()

<b>NAME</b>	<b>m2IcmpInit()</b> – initialize MIB-II ICMP-group access
<b>SYNOPSIS</b>	<b>STATUS</b> m2IcmpInit (void)
<b>DESCRIPTION</b>	This routine allocates the resources needed to allow access to the MIB-II ICMP-group variables. This routine must be called before any ICMP variables can be accessed.
<b>RETURNS</b>	OK, always.
<b>SEE ALSO</b>	m2IcmpLib, m2IcmpGroupInfoGet(), m2IcmpDelete()

---

## m2If8023PacketCount()

<b>NAME</b>	<b>m2If8023PacketCount()</b> – increment the packet counters for an 802.3 device
<b>SYNOPSIS</b>	<b>STATUS</b> m2If8023PacketCount ( M2_ID * pId,                  /* The pointer to the device M2_ID object */ UINT  ctrl,                  /* Update In or Out counters */ UCHAR * pPkt,                /* The incoming/outgoing packet */ ULONG  pktLen                /* Length of the packet */ )
<b>DESCRIPTION</b>	This function is used to update basic interface counters for a packet. The <i>ctrl</i> argument specifies whether the packet is being sent or just received (M2_PACKET_IN or M2_PACKET_OUT). This function only works for 802.3 devices as it understand the Ethernet packet format. The following counters are updated:  - ifInOctets - ifInUcastPkts - ifInNUcastPkts - ifOutOctets - ifOutUcastPkts - ifOutNUcastPkts - ifInMulticastPkts - ifInBroadcastPkts

- ifOutMulticastPkts
- ifOutBroadcastPkts
- ifHCInOctets
- ifHCInUcastPkts
- ifHCOctets
- ifHCOUcastPkts
- ifHCInMulticastPkts
- ifHCInBroadcastPkts
- ifHCOOutMulticastPkts
- ifHCOOutBroadcastPkts
- ifCounterDiscontinuityTime

This function should be called right after the **netMblkToBufCopy()** function has been completed. The first 6 bytes in the resulting buffer must contain the destination MAC address and the second 6 bytes of the buffer must contain the source MAC address.

The type of MAC address (*i.e.*, broadcast, multicast, or unicast) is determined by the following:

broadcast address: ff:ff:ff:ff:ff:ff

multicast address: first bit is set

unicast address: any other address not matching the above

**RETURNS** ERROR, if the M2\_ID is NULL, or the ctrl is invalid; OK, if the counters were updated.

**SEE ALSO** m2IfLib

---

## m2IfAlloc()

**NAME** m2IfAlloc() – allocate the structure for the interface table

**SYNOPSIS**

```
M2_ID * m2IfAlloc
(
    ULONG    ifType,           /* If type of the interface */
    UCHAR * pEnetAddr,        /* Physical address of interface */
    ULONG    addrLen,         /* Address length */
    ULONG    mtuSize,         /* MTU of interface */
    ULONG    speed,           /* Speed of the interface */
    char *   pName,           /* Name of the device */
    int      unit             /* Unit number of the device */
)
```

- DESCRIPTION** This routine is called by the driver during initialization of the interface. The memory for the interface table is allocated here. We also set the default update routines in the **M2\_ID** struct. These fields can later be overloaded using the installed routines in the **M2\_ID**. Once this function returns, it is the driver's responsibility to set the **pMib2Tbl** pointer in the **END** object to the new **M2\_ID**.
- When this call returns, the calling routine must set the **END\_MIB\_2233** bit of the flags field in the **END** object.
- RETURNS** Pointer to the **M2\_ID** structure that was allocated.
- SEE ALSO** **m2IfLib**

---

## m2IfCommonValsGet()

- NAME** **m2IfCommonValsGet()** – get the common values
- SYNOPSIS**
- ```
void m2IfCommonValsGet
(
    M2_DATA *    pM2Data,      /* The requested struct */
    M2_IFINDEX * pIfIndexEntry /* The ifindex node */
)
```
- DESCRIPTION** This function updates the requested struct with all the data that is independent of the driver ioctl. This information can be obtained from the ifnet structures.
- RETURNS** n/a
- SEE ALSO** **m2IfLib**

---

## m2IfCounterUpdate()

**NAME** `m2IfCounterUpdate()` – increment interface counters

**SYNOPSIS**

```
STATUS m2IfCounterUpdate
(
    M2_ID * pId,           /* The pointer to the device M2_ID object */
    UINT   ctrId,         /* Counter to update */
    ULONG  value          /* Amount to update the counter by */
)
```

**DESCRIPTION** This function is used to directly update an interface counter. The counter is specified by *ctrId* and the amount to increment it is specified by *value*. If the counter would roll over then the `ifCounterDiscontinuityTime` is updated with the current system uptime.

**RETURNS** ERROR if the `M2_ID` is NULL, OK if the counter was updated.

**SEE ALSO** `m2IfLib`

**M**

---

## m2IfCtrUpdateRtnInstall()

**NAME** `m2IfCtrUpdateRtnInstall()` – install an interface counter update routine

**SYNOPSIS**

```
STATUS m2IfCtrUpdateRtnInstall
(
    M2_ID *          pId,
    M2_CTR_UPDATE_RTN pRtn
)
```

**DESCRIPTION** This function installs a routine in the `M2_ID`. This routine is able to update a single specified interface counter.

**RETURNS** ERROR if the `M2_ID` is NULL, OK if the routine was installed.

**SEE ALSO** `m2IfLib`

---

## m2IfDefaultValsGet()

|                    |                                                                                                                                                         |
|--------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>        | <b>m2IfDefaultValsGet()</b> – get the default values for the counters                                                                                   |
| <b>SYNOPSIS</b>    | <pre>void m2IfDefaultValsGet (     M2_DATA *    pM2Data,      /* The requested entry */     M2_IFINDEX * pIfIndexEntry /* The ifindex node */ )</pre>   |
| <b>DESCRIPTION</b> | This function fills the given struct with the default values as specified in the RFC. We will enter this routine only if the ioctl to the driver fails. |
| <b>RETURNS</b>     | n/a                                                                                                                                                     |
| <b>SEE ALSO</b>    | <b>m2IfLib</b>                                                                                                                                          |

---

## m2IfDelete()

|                    |                                                                                                                                                                      |
|--------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>        | <b>m2IfDelete()</b> – delete all resources used to access the interface group                                                                                        |
| <b>SYNOPSIS</b>    | <pre>STATUS m2IfDelete (void)</pre>                                                                                                                                  |
| <b>DESCRIPTION</b> | This routine frees all the resources allocated at the time the group was initialized. The interface group should not be accessed after this routine has been called. |
| <b>RETURNS</b>     | OK, always.                                                                                                                                                          |
| <b>SEE ALSO</b>    | <b>m2IfLib</b> , <b>m2IfInit()</b> , <b>m2IfGroupInfoGet()</b> , <b>m2IfTblEntryGet()</b> , <b>m2IfTblEntrySet()</b>                                                 |

---

## m2IfFree()

|                    |                                                                                                                                                                 |
|--------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>        | <b>m2IfFree()</b> – free an interface data structure                                                                                                            |
| <b>SYNOPSIS</b>    | <pre>STATUS m2IfFree (     M2_ID * pId           /* pointer to the driver's M2_ID object */ )</pre>                                                             |
| <b>DESCRIPTION</b> | This routine frees the given <b>M2_ID</b> . Note if the driver is not an RFC 2233 driver then the <b>M2_ID</b> is <b>NULL</b> and this function simply returns. |
| <b>RETURNS</b>     | <b>OK</b> if successful, <b>ERROR</b> otherwise                                                                                                                 |
| <b>SEE ALSO</b>    | <b>m2IfLib</b>                                                                                                                                                  |

---

## m2IfGenericPacketCount()



|                    |                                                                                                                                                                                                                                                                                                                                                                                                                        |
|--------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>        | <b>m2IfGenericPacketCount()</b> – increment the interface packet counters                                                                                                                                                                                                                                                                                                                                              |
| <b>SYNOPSIS</b>    | <pre>STATUS m2IfGenericPacketCount (     M2_ID * pId,           /* The pointer to the device M2_ID object */     UINT  ctrl,           /* Update In or Out counters */     UCHAR * pPkt,         /* The incoming/outgoing packet */     ULONG pktLen          /* Length of the packet */ )</pre>                                                                                                                       |
| <b>DESCRIPTION</b> | This function updates the basic interface counters for a packet. It knows nothing of the underlying media. Thus, so only the <b>ifInOctets</b> , <b>ifHCInOctets</b> , <b>ifOutOctets</b> , <b>ifHCOctets</b> , and <b>ifCounterDiscontinuityTime</b> variables are incremented. The <i>ctrl</i> argument specifies whether the packet is being sent or just received ( <b>M2_PACKET_IN</b> or <b>M2_PACKET_OUT</b> ). |
| <b>RETURNS</b>     | <b>ERROR</b> if the <b>M2_ID</b> is <b>NULL</b> , <b>OK</b> if the counters were updated.                                                                                                                                                                                                                                                                                                                              |
| <b>SEE ALSO</b>    | <b>m2IfLib</b>                                                                                                                                                                                                                                                                                                                                                                                                         |

---

## m2IfGroupInfoGet()

|                    |                                                                                                                                |
|--------------------|--------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>        | <b>m2IfGroupInfoGet()</b> – get the MIB-II interface-group scalar variables                                                    |
| <b>SYNOPSIS</b>    | <pre>STATUS m2IfGroupInfoGet (     M2_INTERFACE * pIfInfo    /* pointer to interface group structure */ )</pre>                |
| <b>DESCRIPTION</b> | This routine fills the interface-group structure at <i>pIfInfo</i> with the values of MIB-II interface-group global variables. |
| <b>RETURNS</b>     | OK, or ERROR if <i>pIfInfo</i> is not a valid pointer.                                                                         |
| <b>ERRNO</b>       | S_m2Lib_INVALID_PARAMETER                                                                                                      |
| <b>SEE ALSO</b>    | m2IfLib, m2IfInit(), m2IfTb1EntryGet(), m2IfTb1EntrySet(), m2IfDelete()                                                        |

---

## m2IfInit()

|                    |                                                                                                                                                                                                                                                                                                                                                                                                              |
|--------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>        | <b>m2IfInit()</b> – initialize MIB-II interface-group routines                                                                                                                                                                                                                                                                                                                                               |
| <b>SYNOPSIS</b>    | <pre>STATUS m2IfInit (     FUNCPtr pTrapRtn,        /* pointer to user trap generator */     void * pTrapArg         /* pointer to user trap generator argument */ )</pre>                                                                                                                                                                                                                                   |
| <b>DESCRIPTION</b> | This routine allocates the resources needed to allow access to the MIB-II interface-group variables. This routine must be called before any interface variables can be accessed. The input parameter <i>pTrapRtn</i> is an optional pointer to a user-supplied SNMP trap generator. The input parameter <i>pTrapArg</i> is an optional argument to the trap generator. Only one trap generator is supported. |

### VXWORKS AE PROTECTION DOMAINS

Under VxWorks AE, you can call **m2IfInit()** from within the kernel protection domain only, and the data referenced in the *pTrapRtn* and *pTrapArg* parameters must reside in the kernel protection domain. This restriction does not apply to non-AE versions of VxWorks.

**RETURNS** OK, if successful; **ERROR**, if an error occurred.

**ERRNO** S\_m2Lib\_CANT\_CREATE\_IF\_SEM

**SEE ALSO** m2IfLib, m2IfGroupInfoGet(), m2IfTblEntryGet(), m2IfTblEntrySet(), m2IfDelete()

---

## m2IfPktCountRtnInstall()

**NAME** m2IfPktCountRtnInstall() – install an interface packet counter routine

**SYNOPSIS**

```
STATUS m2IfPktCountRtnInstall
(
    M2_ID *          pId,
    M2_PKT_COUNT_RTN pRtn
)
```

**DESCRIPTION** This function installs a routine in the M2\_ID. This routine is a packet counter which is able to update all the interface counters.

**RETURNS** **ERROR** if the M2\_ID is NULL, **OK** if the routine was installed.

**SEE ALSO** m2IfLib

---

## m2IfRcvAddrEntryGet()

**NAME** m2IfRcvAddrEntryGet() – get the rcvAddress table entries for a given address

**SYNOPSIS**

```
STATUS m2IfRcvAddrEntryGet
(
    int          search,      /* exact search or next search */
    int *       pIndex,      /* pointer to the ifIndex */
    M2_IFRCVADDRTBL * pIfReqEntry /* struct for the values */
)
```

**DESCRIPTION** This function returns the exact or the next value in the ifRcvAddressTable based on the value of the search parameter. In order to identify the appropriate entry, this function needs two identifiers - the ifIndex of the interface and the physical address for which the status or the type is being requested. For a M2\_EXACT\_VALUE search, this function returns the status and the type of the physical address in the instance. For a M2\_NEXT\_VALUE

**m2IfRcvAddrEntrySet()**

search, it returns the type and status of the lexicographic successor of the physical address seen in the instance.

|                 |                                                                                                                             |
|-----------------|-----------------------------------------------------------------------------------------------------------------------------|
| <b>RETURNS</b>  | OK, or ERROR if the input parameter is not specified, an interface is no longer valid, or the interface index is incorrect. |
| <b>ERRNO</b>    | S_m2Lib_INVALID_PARAMETER<br>S_m2Lib_ENTRY_NOT_FOUND<br>S_m2Lib_IF_CNFG_CHANGED                                             |
| <b>SEE ALSO</b> | m2IfLib                                                                                                                     |

---

## m2IfRcvAddrEntrySet()

|                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
|--------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>        | m2IfRcvAddrEntrySet() – modify the entries of the rcvAddressTable                                                                                                                                                                                                                                                                                                                                                                                                                               |
| <b>SYNOPSIS</b>    | <pre> STATUS m2IfRcvAddrEntrySet (     int          varToSet,    /* entries that need to be modified */     int          index,      /* search type */     M2_IFRCVADRTBL * pIfReqEntry /* struct containing the new values */ ) </pre>                                                                                                                                                                                                                                                         |
| <b>DESCRIPTION</b> | <p>This function modifies the status and type fields of a given receive address associated with a given interface. <i>varToSet</i> identifies the fields for which the change is being requested. We can also add multicast addresses by creating a new row in the table. The physical address is stripped from the instance value of the SNMP request. This routine does not allow the deletion of a unicast address. Neither does it allow the unicast address to be modified or created.</p> |
| <b>RETURNS</b>     | OK, or ERROR if the input parameter is not specified, an interface is no longer valid, the interface index is incorrect, or the <b>ioctl()</b> command to the interface fails.                                                                                                                                                                                                                                                                                                                  |
| <b>ERRNO</b>       | S_m2Lib_INVALID_PARAMETER<br>S_m2Lib_ENTRY_NOT_FOUND<br>S_m2Lib_IF_CNFG_CHANGED                                                                                                                                                                                                                                                                                                                                                                                                                 |
| <b>SEE ALSO</b>    | m2IfLib, m2IfInit(), m2IfGroupInfoGet(), m2IfTblEntryGet(), m2IfDelete()                                                                                                                                                                                                                                                                                                                                                                                                                        |

---

## m2IfStackEntryGet()

**NAME** m2IfStackEntryGet() – get a MIB-II interface-group table entry

**SYNOPSIS**

```
STATUS m2IfStackEntryGet
(
    int          search,      /* M2_EXACT_VALUE or M2_NEXT_VALUE */
    int *        pHighIndex, /* the higher layer's ifIndex */
    M2_IFSTACKTBL * pIfReqEntry /* pointer to the requested entry */
)
```

**DESCRIPTION** This routine maps the given high and low indexes to the interfaces in the AVL tree. Using the *high* and *low* indexes, we retrieve the nodes in question and walk through their linked lists to get to the right relation. Once we get to the correct node, we can return the values based on the M2\_EXACT\_VALUE and the M2\_NEXT\_VALUE searches.

**RETURNS** OK, or ERROR if the input parameter is not specified, or a match is not found.

**ERRNO** S\_m2Lib\_INVALID\_PARAMETER  
S\_m2Lib\_ENTRY\_NOT\_FOUND

**SEE ALSO** m2IfLib

---

## m2IfStackEntrySet()

**NAME** m2IfStackEntrySet() – modify the status of a relationship

**SYNOPSIS**

```
STATUS m2IfStackEntrySet
(
    int          highIndex, /* The higher layer's ifIndex */
    M2_IFSTACKTBL * pIfReqEntry /* The requested entry */
)
```

**DESCRIPTION** This routine selects the interfaces specified in the input parameters *pIfReqEntry* and *highIndex* and sets the interface's status to the requested state.

**RETURNS** OK, or ERROR if the input parameter is not specified, an interface is no longer valid, or the interface index is incorrect.

**ERRNO** S\_m2Lib\_INVALID\_PARAMETER  
S\_m2Lib\_ENTRY\_NOT\_FOUND  
S\_m2Lib\_IF\_CNFG\_CHANGED

**SEE ALSO** m2IfLib

---

## m2IfStackTblUpdate()

**NAME** m2IfStackTblUpdate() – update the relationship between the sub-layers

**SYNOPSIS**

```
STATUS m2IfStackTblUpdate
(
    UINT lowerIndex,          /* The ifIndex of the lower sub-layer */
    UINT higherIndex,       /* The ifIndex of the higher sub-layer */
    int  action              /* insert or remove */
)
```

**DESCRIPTION** This function must be called to setup the relationship between the ifIndex values for each sub-layer. This information is required to support the ifStackTable for RFC 2233. Using this data, we can easily determine which sub-layer runs on top of which other.

*action* is either M2\_STACK\_TABLE\_INSERT or M2\_STACK\_TABLE\_REMOVE.

Each AVL node keeps a linked list of all the layers that are directly beneath it. Thus by walking through the AVL nodes in an orderly way, we can understand the relationships between all the interfaces.

**RETURNS** OK upon successful addition  
ERROR otherwise.

**SEE ALSO** m2IfLib

---

## m2IfTableUpdate()

**NAME** m2IfTableUpdate() – insert or remove an entry in the ifTable

**SYNOPSIS**

```
STATUS m2IfTableUpdate
(
    struct ifNet * pIfNet,
    UINT          status,      /* attaching or detaching */
    int (* if_ioctl) (struct socket*,u_long,caddr_t),
                    /* protocol-specific ioctl or null for default (ethernet) */
    STATUS (* addr_get) (struct ifnet* , M2_IFINDEX* )
                    /* func to grab the interface's addrs, null */
                    /* for default (ethernet) */
)
```

**DESCRIPTION** This routine is called by **if\_attach** and **if\_detach** to insert/remove an entry from the local **m2IfLib** ifTable. The status can be either **M2\_IF\_TABLE\_INSERT** or **M2\_IF\_TABLE\_REMOVE**. The ifIndex that is searched for in the AVL tree is specified in given the ifnet struct. *if\_ioctl* is a function pointer to change the flags on the interface. *addr\_get* is a function pointer to add the interface's addresses to ifRcvAddressTable. Ethernet interfaces can use NULL for both function pointers, other interfaces will need to pass an appropriate function.

**RETURNS** ERROR if entry does not exist, OK if the entry was deleted

**SEE ALSO** m2IfLib

---

## m2IfTblEntryGet()

**NAME** m2IfTblEntryGet() – get a MIB-II interface-group table entry

**SYNOPSIS**

```
STATUS m2IfTblEntryGet
(
    int          search,      /* M2_EXACT_VALUE or M2_NEXT_VALUE */
    void * pIfReqEntry      /* pointer to requested interface entry */
)
```

**DESCRIPTION** This routine maps the MIB-II interface index to the system's internal interface index. The internal representation is in the form of a balanced AVL tree indexed by ifIndex of the interface. The *search* parameter is set to either **M2\_EXACT\_VALUE** or **M2\_NEXT\_VALUE**; for

**m2IfTblEntrySet()**

a discussion of its use, see the manual entry for **m2Lib**. The interface table values are returned in a structure of type **M2\_DATA**, which is passed as the second argument to this routine.

|                 |                                                                                 |
|-----------------|---------------------------------------------------------------------------------|
| <b>RETURNS</b>  | OK, or ERROR if the input parameter is not specified, or a match is not found.  |
| <b>ERRNO</b>    | S_m2Lib_INVALID_PARAMETER<br>S_m2Lib_ENTRY_NOT_FOUND                            |
| <b>SEE ALSO</b> | m2IfLib, m2Lib, m2IfInit(), m2IfGroupInfoGet(), m2IfTblEntrySet(), m2IfDelete() |

---

## m2IfTblEntrySet()

|                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
|--------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>        | m2IfTblEntrySet() – set the state of a MIB-II interface entry to UP or DOWN                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| <b>SYNOPSIS</b>    | <pre>STATUS m2IfTblEntrySet (     void * pIfReqEntry      /* pointer to requested entry to change */ )</pre>                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
| <b>DESCRIPTION</b> | <p>This routine selects the interface specified in the input parameter <i>pIfReqEntry</i> and sets the interface parameters to the requested state. It is the responsibility of the calling routine to set the interface index, and to make sure that the state specified in the <b>ifAdminStatus</b> field of the structure at <i>pIfTblEntry</i> is a valid MIB-II state, up(1) or down(2).</p> <p>The fields that can be modified by this routine are the following: <b>ifAdminStatus</b>, <b>ifAlias</b>, <b>ifLinkUpDownTrapEnable</b> and <b>ifName</b>.</p> |
| <b>RETURNS</b>     | OK, or ERROR if the input parameter is not specified, an interface is no longer valid, the interface index is incorrect, or the <b>ioctl()</b> command to the interface fails.                                                                                                                                                                                                                                                                                                                                                                                     |
| <b>ERRNO</b>       | S_m2Lib_INVALID_PARAMETER<br>S_m2Lib_ENTRY_NOT_FOUND<br>S_m2Lib_IF_CNFG_CHANGED                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| <b>SEE ALSO</b>    | m2IfLib, m2IfInit(), m2IfGroupInfoGet(), m2IfTblEntryGet(), m2IfDelete()                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |

## m2IfVariableUpdate()

**NAME** m2IfVariableUpdate() – update the contents of an interface non-counter object

**SYNOPSIS**

```

STATUS m2IfVariableUpdate
(
    M2_ID * pId,           /* The pointer to the device M2_ID object */
    UINT   varId,         /* Variable to update */
    caddr_t pData         /* Data to use */
)

```

**DESCRIPTION** This function is used to update an interface variable. The variable is specified by `varId` and the data to use is specified by `pData`. Note that different variable expect different types of data. Here is a list of the variables and the type of data expected. Therefore, `pData` will be cast to the type listed below for each variable.

| Variable               | Cast to Type  |
|------------------------|---------------|
| ifDescr                | char *        |
| ifType                 | UINT          |
| ifMtu                  | ULONG         |
| ifSpeed                | ULONG         |
| ifPhysAddress          | M2_PHYADDR *  |
| ifAdminStatus          | ULONG         |
| ifOperStatus           | ULONG         |
| ifLastChange           | ULONG         |
| ifOutQLen              | ULONG         |
| ifSpecific             | M2_OBJECTID * |
| ifName                 | char *        |
| ifLinkUpDownTrapEnable | UINT          |
| ifHighSpeed            | ULONG         |
| ifPromiscuousMode      | UINT          |
| ifConnectorPresent     | UINT          |
| ifAlias                | char *        |

**RETURNS** ERROR, if the M2\_ID is NULL; OK, if the variable was updated.

**SEE ALSO** m2IfLib



---

## m2IfVarUpdateRtnInstall()

**NAME** m2IfVarUpdateRtnInstall() – install an interface variable update routine

**SYNOPSIS**

```
STATUS m2IfVarUpdateRtnInstall
(
    M2_ID *          pId,
    M2_VAR_UPDATE_RTN pRtn
)
```

**DESCRIPTION** This function installs a routine in the M2\_ID. This routine is able to update a single specified interface variable.

**RETURNS** ERROR if the M2\_ID is NULL, OK if the routine was installed.

**SEE ALSO** m2IfLib

---

## m2Init()

**NAME** m2Init() – initialize the SNMP MIB-2 library

**SYNOPSIS**

```
STATUS m2Init
(
    char *          pMib2SysDescr, /* sysDescr */
    char *          pMib2SysContact, /* sysContact */
    char *          pMib2SysLocation, /* sysLocation */
    M2_OBJECTID *  pMib2SysObjectId, /* sysObjectId */
    FUNCPTR        pTrapRtn, /* link up/down -trap routine */
    void *          pTrapArg, /* trap routine arg */
    int             maxRouteTableSize /* max size of routing table */
)
```

**DESCRIPTION** This routine initializes the MIB-2 library by calling the initialization routines for each MIB-2 group. The parameters *pMib2SysDescr*, *pMib2SysContact*, *pMib2SysLocation*, and *pMib2SysObjectId* are passed directly to **m2SysInit()**; *pTrapRtn* and *pTrapArg* are passed directly to **m2IfInit()**; and *maxRouteTableSize* is passed to **m2IpInit()**.

**RETURNS** OK if successful, otherwise ERROR.

**SEE ALSO** m2Lib, m2SysInit(), m2TcpInit(), m2UdpInit(), m2IcmpInit(), m2IfInit(), m2IpInit()

---

## m2IpAddrTblEntryGet()

**NAME** `m2IpAddrTblEntryGet()` – get an IP MIB-II address entry

**SYNOPSIS**

```
STATUS m2IpAddrTblEntryGet
(
    int          search,          /* M2_EXACT_VALUE or M2_NEXT_VALUE */
    M2_IPADDR_TBL * pIpAddrTblEntry /* ptr to requested IP address entry */
)
```

**DESCRIPTION** This routine traverses the IP address table and does an `M2_EXACT_VALUE` or a `M2_NEXT_VALUE` search based on the `search` parameter. The calling routine is responsible for supplying a valid MIB-II entry index in the input structure `pIpAddrTblEntry`. The index is the local IP address. The first entry in the table is retrieved by doing a NEXT search with the index field set to zero.

**RETURNS** OK, ERROR if the input parameter is not specified, or a match is not found.

**ERRNO** `S_m2Lib_INVALID_PARAMETER`  
`S_m2Lib_ENTRY_NOT_FOUND`

**SEE ALSO** `m2IpLib`, `m2Lib`, `m2IpInit()`, `m2IpGroupInfoGet()`, `m2IpGroupInfoSet()`, `m2IpAtransTblEntrySet()`, `m2IpRouteTblEntryGet()`, `m2IpRouteTblEntrySet()`, `m2IpDelete()`

---

## m2IpAtransTblEntryGet()

**NAME** `m2IpAtransTblEntryGet()` – get a MIB-II ARP table entry

**SYNOPSIS**

```
STATUS m2IpAtransTblEntryGet
(
    int          search,          /* M2_EXACT_VALUE or M2_NEXT_VALUE */
    M2_IPATRANSTBL * pReqIpAtEntry /* ptr to the requested ARP entry */
)
```

**DESCRIPTION** This routine traverses the ARP table and does an `M2_EXACT_VALUE` or a `M2_NEXT_VALUE` search based on the `search` parameter. The calling routine is responsible for supplying a valid MIB-II entry index in the input structure `pReqIpAtEntry`. The index is made up of the network interface index and the IP address corresponding to the physical

address. The first entry in the table is retrieved by doing a NEXT search with the index fields set to zero.

- RETURNS** OK, ERROR if the input parameter is not specified, or a match is not found.
- ERRNO** S\_m2Lib\_INVALID\_PARAMETER  
S\_m2Lib\_ENTRY\_NOT\_FOUND
- SEE ALSO** m2IpLib, m2Lib, m2IpInit(), m2IpGroupInfoGet(), m2IpGroupInfoSet(), m2IpAtransTblEntrySet(), m2IpRouteTblEntryGet(), m2IpRouteTblEntrySet(), m2IpDelete()

---

## m2IpAtransTblEntrySet()

- NAME** m2IpAtransTblEntrySet() – add, modify, or delete a MIB-II ARP entry
- SYNOPSIS**
- ```
STATUS m2IpAtransTblEntrySet
(
    M2_IPATRANSTBL * pReqIpAtEntry /* pointer to MIB-II ARP entry */
)
```
- DESCRIPTION** This routine traverses the ARP table for the entry specified in the parameter *pReqIpAtEntry*. An ARP entry can be added, modified, or deleted. A MIB-II entry index is specified by the destination IP address and the physical media address. A new ARP entry can be added by specifying all the fields in the parameter *pReqIpAtEntry*. An entry can be modified by specifying the MIB-II index and the field that is to be modified. An entry is deleted by specifying the index and setting the type field in the input parameter *pReqIpAtEntry* to the MIB-II value “invalid” (2).
- RETURNS** OK, or ERROR if the input parameter is not specified, the physical address is not specified for an add/modify request, or the **ioctl()** request to the ARP module fails.
- ERRNO** S\_m2Lib\_INVALID\_PARAMETER  
S\_m2Lib\_ARP\_PHYSADDR\_NOT\_SPECIFIED
- SEE ALSO** m2IpLib, m2IpInit(), m2IpGroupInfoGet(), m2IpGroupInfoSet(), m2IpAddrTblEntryGet(), m2IpRouteTblEntryGet(), m2IpRouteTblEntrySet(), m2IpDelete()

---

## m2IpDelete( )

**NAME** m2IpDelete( ) – delete all resources used to access the IP group

**SYNOPSIS** `STATUS m2IpDelete (void)`

**DESCRIPTION** This routine frees all the resources allocated when the IP group was initialized. The IP group should not be accessed after this routine has been called.

**RETURNS** OK, always.

**SEE ALSO** m2IpLib, m2IpInit( ), m2IpGroupInfoGet( ), m2IpGroupInfoSet( ), m2IpAddrTblEntryGet( ), m2IpAtransTblEntrySet( ), m2IpRouteTblEntryGet( ), m2IpRouteTblEntrySet( )

---

## m2IpGroupInfoGet( )

**NAME** m2IpGroupInfoGet( ) – get the MIB-II IP-group scalar variables

**SYNOPSIS** `STATUS m2IpGroupInfoGet  
(   
M2_IP * pIpInfo /* pointer to IP MIB-II global group variables */  
)`

**DESCRIPTION** This routine fills in the IP structure at *pIpInfo* with the values of MIB-II IP global variables.

**RETURNS** OK, or ERROR if *pIpInfo* is not a valid pointer.

**ERRNO** S\_m2Lib\_INVALID\_PARAMETER

**SEE ALSO** m2IpLib, m2IpInit( ), m2IpGroupInfoSet( ), m2IpAddrTblEntryGet( ), m2IpAtransTblEntrySet( ), m2IpRouteTblEntryGet( ), m2IpRouteTblEntrySet( ), m2IpDelete( )

**M**

---

## m2IpGroupInfoSet()

<b>NAME</b>	<b>m2IpGroupInfoSet()</b> – set MIB-II IP-group variables to new values
<b>SYNOPSIS</b>	<pre><b>STATUS</b> m2IpGroupInfoSet (     unsigned int varToSet, /* bit field used to set variables */     M2_IP *      pIpInfo   /* ptr to the MIB-II IP group global variables */ )</pre>
<b>DESCRIPTION</b>	This routine sets one or more variables in the IP group, as specified in the input structure <i>pIpInfo</i> and the bit field parameter <i>varToSet</i> .
<b>RETURNS</b>	OK, or ERROR if <i>pIpInfo</i> is not a valid pointer, or <i>varToSet</i> has an invalid bit field.
<b>ERRNO</b>	S_m2Lib_INVALID_PARAMETER S_m2Lib_INVALID_VAR_TO_SET
<b>SEE ALSO</b>	m2IpLib, m2IpInit(), m2IpGroupInfoGet(), m2IpAddrTblEntryGet(), m2IpAtransTblEntrySet(), m2IpRouteTblEntryGet(), m2IpRouteTblEntrySet(), m2IpDelete()

---

## m2IpInit()

<b>NAME</b>	<b>m2IpInit()</b> – initialize MIB-II IP-group access
<b>SYNOPSIS</b>	<pre><b>STATUS</b> m2IpInit (     int maxRouteTableSize /* max size of routing table */ )</pre>
<b>DESCRIPTION</b>	This routine allocates the resources needed to allow access to the MIB-II IP variables. This routine must be called before any IP variables can be accessed. The parameter <i>maxRouteTableSize</i> is used to increase the default size of the MIB-II route table cache.
<b>RETURNS</b>	OK, or ERROR if the route table or the route semaphore cannot be allocated.
<b>ERRNO</b>	S_m2Lib_CANT_CREATE_ROUTE_SEM

SEE ALSO m2IpLib, m2IpGroupInfoGet(), m2IpGroupInfoSet(), m2IpAddrTblEntryGet(), m2IpAtransTblEntrySet(), m2IpRouteTblEntryGet(), m2IpRouteTblEntrySet(), m2IpDelete()

---

## m2IpRouteTblEntryGet()

NAME m2IpRouteTblEntryGet() – get a MIB-2 routing table entry

SYNOPSIS **STATUS** m2IpRouteTblEntryGet  

```
(
    int          search,          /* M2_EXACT_VALUE or M2_NEXT_VALUE */
    M2_IPROUTETBL * pIpRouteTblEntry /* route table entry */
)
```

DESCRIPTION This routine retrieves MIB-II information about an entry in the network routing table and returns it in the caller-supplied structure *pIpRouteTblEntry*.

The routine compares routing table entries to the address specified by the **ipRouteDest** member of the *pIpRouteTblEntry* structure, and retrieves an entry chosen by the *search* type (M2\_EXACT\_VALUE or M2\_NEXT\_VALUE, as described in the manual entry for **m2Lib**).

RETURNS OK if successful, otherwise **ERROR**.

ERRNO S\_m2Lib\_INVALID\_PARAMETER  
 S\_m2Lib\_ENTRY\_NOT\_FOUND

SEE ALSO m2IpLib, m2Lib, m2IpInit(), m2IpGroupInfoGet(), m2IpGroupInfoSet(), m2IpAddrTblEntryGet(), m2IpRouteTblEntryGet(), m2IpRouteTblEntrySet(), m2IpDelete()

---

## m2IpRouteTblEntrySet()

NAME m2IpRouteTblEntrySet() – set a MIB-II routing table entry

SYNOPSIS **STATUS** m2IpRouteTblEntrySet  

```
(
    int          varToSet,        /* variable to set */
    M2_IPROUTETBL * pIpRouteTblEntry /* route table entry */
)
```

**m2RipDelete()**

<b>DESCRIPTION</b>	<p>This routine adds, changes, or deletes a network routing table entry. The table entry to be modified is specified by the <b>ipRouteDest</b> and <b>ipRouteNextHop</b> members of the <i>pIpRouteTblEntry</i> structure.</p> <p>The <i>varToSet</i> parameter is a bit-field mask that specifies which values in the route table entry are to be set.</p> <p>If <i>varToSet</i> has the <b>M2_IP_ROUTE_TYPE</b> bit set and <b>ipRouteType</b> has the value of <b>M2_ROUTE_TYPE_INVALID</b>, then the routing table entry is deleted.</p> <p>If <i>varToSet</i> has either the <b>M2_IP_ROUTE_DEST</b>, <b>M2_IP_ROUTE_NEXT_HOP</b> and the <b>M2_IP_ROUTE_MASK</b> bits set, then a new route entry is added to the table.</p>
<b>RETURNS</b>	OK if successful, otherwise <b>ERROR</b> .
<b>SEE ALSO</b>	<b>m2IpLib</b> , <b>m2IpInit()</b> , <b>m2IpGroupInfoGet()</b> , <b>m2IpGroupInfoSet()</b> , <b>m2IpAddrTblEntryGet()</b> , <b>m2IpRouteTblEntryGet()</b> , <b>m2IpRouteTblEntrySet()</b> , <b>m2IpDelete()</b>

---

## m2RipDelete()

<b>NAME</b>	<b>m2RipDelete()</b> – delete the RIP MIB support
<b>SYNOPSIS</b>	<b>STATUS</b> <b>m2RipDelete</b> (void)
<b>DESCRIPTION</b>	This routine should be called after all <b>m2RipLib</b> calls are completed.
<b>RETURNS</b>	OK, always.
<b>SEE ALSO</b>	<b>m2RipLib</b>

---

## m2RipGlobalCountersGet()

<b>NAME</b>	m2RipGlobalCountersGet() – get MIB-II RIP-group global counters
<b>SYNOPSIS</b>	<pre> STATUS m2RipGlobalCountersGet (     M2_RIP2_GLOBAL_GROUP* pRipGlobal ) </pre>
<b>DESCRIPTION</b>	This routine fills in an M2_RIP2_GLOBAL_GROUP structure pointed to by <i>pRipGlobal</i> with the values of the MIB-II RIP-group global counters.
<b>RETURNS</b>	OK or ERROR.
<b>ERRNO</b>	S_m2Lib_INVALID_PARAMETER
<b>SEE ALSO</b>	m2RipLib, m2RipInit()

---

## m2RipIfConfEntryGet()

<b>NAME</b>	m2RipIfConfEntryGet() – get MIB-II RIP-group interface entry
<b>SYNOPSIS</b>	<pre> STATUS m2RipIfConfEntryGet (     int                search,     M2_RIP2_IFCONF_ENTRY* pRipIfConf ) </pre>
<b>DESCRIPTION</b>	<p>This routine retrieves the interface configuration for the interface serving the subnet of the IP address contained in the M2_RIP2_IFCONF_ENTRY structure passed to it. <i>pRipIfConf</i> is a pointer to an M2_RIP2_IFCONF_ENTRY structure which the routine will fill in upon successful completion.</p> <p>This routine either returns an exact match if <i>search</i> is M2_EXACT_VALUE, or the next value greater than or equal to the value supplied if the <i>search</i> is M2_NEXT_VALUE.</p>
<b>RETURNS</b>	OK, or ERROR if <i>pRipIfConf</i> was invalid or the interface was not found.
<b>ERRNO</b>	S_m2Lib_INVALID_PARAMETER S_m2Lib_ENTRY_NOT_FOUND
<b>SEE ALSO</b>	m2RipLib, m2RipInit()

---

## m2RipIfConfEntrySet()

**NAME** m2RipIfConfEntrySet() – set MIB-II RIP-group interface entry

**SYNOPSIS**

```
STATUS m2RipIfConfEntrySet
(
    unsigned int      varToSet,
    M2_RIP2_IFCONF_ENTRY* pRipIfConf
)
```

**DESCRIPTION** This routine sets the interface configuration for the interface serving the subnet of the IP address contained in the `M2_RIP2_IFCONF_ENTRY` structure.

*pRipIfConf* is a pointer to an `M2_RIP2_IFCONF_ENTRY` structure which the routine places into the system based on the *varToSet* value.

**RETURNS** OK, or ERROR if *pRipIfConf* is invalid or the interface cannot be found.

**ERRNO** S\_m2Lib\_INVALID\_PARAMETER  
S\_m2Lib\_ENTRY\_NOT\_FOUND

**SEE ALSO** m2RipLib, m2RipInit()

---

## m2RipIfStatEntryGet()

**NAME** m2RipIfStatEntryGet() – get MIB-II RIP-group interface entry

**SYNOPSIS**

```
STATUS m2RipIfStatEntryGet
(
    int      search,
    M2_RIP2_IFSTAT_ENTRY* pRipIfStat
)
```

**DESCRIPTION** This routine retrieves the interface statistics for the interface serving the subnet of the IP address contained in the `M2_RIP2_IFSTAT_ENTRY` structure. *pRipIfStat* is a pointer to an `M2_RIP2_IFSTAT_ENTRY` structure which the routine will fill in upon successful completion.

This routine either returns an exact match if *search* is `M2_EXACT_VALUE`, or the next value greater than or equal to the value supplied if the *search* is `M2_NEXT_VALUE`.

**RETURNS** OK, or **ERROR** if either *pRipIfStat* is invalid or an exact match failed.

**ERRNO** S\_m2Lib\_INVALID\_PARAMETER  
S\_m2Lib\_ENTRY\_NOT\_FOUND

**SEE ALSO** m2RipLib, m2RipInit()

---

## m2RipInit()

**NAME** m2RipInit() – initialize the RIP MIB support

**SYNOPSIS** **STATUS** m2RipInit (void)

**DESCRIPTION** This routine sets up the RIP MIB and should be called before any other **m2RipLib** routine.

**RETURNS** OK, always.

**SEE ALSO** m2RipLib

---

## m2SysDelete()

**NAME** m2SysDelete() – delete resources used to access the MIB-II system group

**SYNOPSIS** **STATUS** m2SysDelete (void)

**DESCRIPTION** This routine frees all the resources allocated at the time the group was initialized. Do not access the system group after calling this routine.

**RETURNS** OK, always.

**SEE ALSO** m2SysLib, m2SysInit(), m2SysGroupInfoGet(), m2SysGroupInfoSet().

**M**

## m2SysGroupInfoGet()

<b>NAME</b>	<b>m2SysGroupInfoGet()</b> – get system-group MIB-II variables
<b>SYNOPSIS</b>	<pre><b>STATUS</b> m2SysGroupInfoGet (     <b>M2_SYSTEM</b> * <b>pSysInfo</b>      /* pointer to MIB-II system group structure */ )</pre>
<b>DESCRIPTION</b>	This routine fills in the structure at <i>pSysInfo</i> with the values of MIB-II system-group variables.
<b>RETURNS</b>	OK, or ERROR if <i>pSysInfo</i> is not a valid pointer.
<b>ERRNO</b>	S_m2Lib_INVALID_PARAMETER
<b>SEE ALSO</b>	m2SysLib, m2SysInit(), m2SysGroupInfoSet(), m2SysDelete()

---

## m2SysGroupInfoSet()

<b>NAME</b>	<b>m2SysGroupInfoSet()</b> – set system-group MIB-II variables to new values
<b>SYNOPSIS</b>	<pre><b>STATUS</b> m2SysGroupInfoSet (     <b>unsigned int</b> <b>varToSet</b>,      /* bit field of variables to set */     <b>M2_SYSTEM</b> * <b>pSysInfo</b>      /* pointer to the system structure */ )</pre>
<b>DESCRIPTION</b>	This routine sets one or more variables in the system group as specified in the input structure at <i>pSysInfo</i> and the bit field parameter <i>varToSet</i> .
<b>RETURNS</b>	OK, or ERROR if <i>pSysInfo</i> is not a valid pointer, or <i>varToSet</i> has an invalid bit field.
<b>ERRNO</b>	S_m2Lib_INVALID_PARAMETER S_m2Lib_INVALID_VAR_TO_SET
<b>SEE ALSO</b>	m2SysLib, m2SysInit(), m2SysGroupInfoGet(), m2SysDelete()

---

## m2SysInit()

**NAME** m2SysInit() – initialize MIB-II system-group routines

**SYNOPSIS**

```
STATUS m2SysInit
(
    char *      pMib2SysDescr,    /* pointer to MIB-2 sysDescr */
    char *      pMib2SysContact, /* pointer to MIB-2 sysContact */
    char *      pMib2SysLocation, /* pointer to MIB-2 sysLocation */
    M2_OBJECTID * pObjectId      /* pointer to MIB-2 ObjectId */
)
```

**DESCRIPTION** This routine allocates the resources needed to allow access to the system-group MIB-II variables. This routine must be called before any system-group variables can be accessed. The input parameters *pMib2SysDescr*, *pMib2SysContact*, *pMib2SysLocation*, and *pObjectId* are optional. The parameters *pMib2SysDescr*, *pObjectId* are read only, as specified by MIB-II, and can be set only by this routine.

**RETURNS** OK, always.

**ERRNO** S\_m2Lib\_CANT\_CREATE\_SYS\_SEM

**SEE ALSO** m2SysLib, m2SysGroupInfoGet(), m2SysGroupInfoSet(), m2SysDelete()

---

## m2TcpConnEntryGet()

**NAME** m2TcpConnEntryGet() – get a MIB-II TCP connection table entry

**SYNOPSIS**

```
STATUS m2TcpConnEntryGet
(
    int          search,          /* M2_EXACT_VALUE or M2_NEXT_VALUE */
    M2_TCPCONNTBL * pReqTcpConnEntry /* input = Index, Output = Entry */
)
```

**DESCRIPTION** This routine traverses the TCP table of users and does an M2\_EXACT\_VALUE or a M2\_NEXT\_VALUE search based on the *search* parameter (see **m2Lib**). The calling routine is responsible for supplying a valid MIB-II entry index in the input structure *pReqTcpConnEntry*. The index is made up of the local IP address, the local port number, the remote IP address, and the remote port. The first entry in the table is retrieved by doing a M2\_NEXT\_VALUE search with the index fields set to zero.

**m2TcpConnEntrySet()**

<b>RETURNS</b>	OK, or ERROR if the input parameter is not specified or a match is not found.
<b>ERRNO</b>	S_m2Lib_INVALID_PARAMETER S_m2Lib_ENTRY_NOT_FOUND
<b>SEE ALSO</b>	m2TcpLib, m2Lib, m2TcpInit(), m2TcpGroupInfoGet(), m2TcpConnEntrySet(), m2TcpDelete()

---

## m2TcpConnEntrySet()

<b>NAME</b>	m2TcpConnEntrySet() – set a TCP connection to the closed state
<b>SYNOPSIS</b>	<pre> <b>STATUS</b> m2TcpConnEntrySet (     M2_TCPCONNTBL * pReqTcpConnEntry /* pointer to TCP connection to close */ ) </pre>
<b>DESCRIPTION</b>	This routine traverses the TCP connection table and searches for the connection specified by the input parameter <i>pReqTcpConnEntry</i> . The calling routine is responsible for providing a valid index as the input parameter <i>pReqTcpConnEntry</i> . The index is made up of the local IP address, the local port number, the remote IP address, and the remote port. This call can only succeed if the connection is in the MIB-II state "deleteTCB" (12). If a match is found, the socket associated with the TCP connection is closed.
<b>RETURNS</b>	OK, or ERROR if the input parameter is invalid, the state of the connection specified at <i>pReqTcpConnEntry</i> is not "closed," the specified connection is not found, a socket is not associated with the connection, or the <b>close()</b> call fails.
<b>SEE ALSO</b>	m2TcpLib, m2TcpInit(), m2TcpGroupInfoGet(), m2TcpConnEntryGet(), m2TcpDelete()

---

## m2TcpDelete()

<b>NAME</b>	m2TcpDelete() – delete all resources used to access the TCP group
<b>SYNOPSIS</b>	<pre> <b>STATUS</b> m2TcpDelete (void) </pre>
<b>DESCRIPTION</b>	This routine frees all the resources allocated at the time the group was initialized. The TCP group should not be accessed after this routine has been called.

**RETURNS** OK, always.

**SEE ALSO** **m2TcpLib**, **m2TcpInit()**, **m2TcpGroupInfoGet()**, **m2TcpConnEntryGet()**, **m2TcpConnEntrySet()**

---

## m2TcpGroupInfoGet()

**NAME** **m2TcpGroupInfoGet()** – get MIB-II TCP-group scalar variables

**SYNOPSIS**

```
STATUS m2TcpGroupInfoGet  
(  
    M2_TCPINFO * pTcpInfo    /* pointer to the TCP group structure */  
)
```

**DESCRIPTION** This routine fills in the TCP structure pointed to by *pTcpInfo* with the values of MIB-II TCP-group scalar variables.

**RETURNS** OK, or **ERROR** if *pTcpInfo* is not a valid pointer.

**ERRNO** **S\_m2Lib\_INVALID\_PARAMETER**

**SEE ALSO** **m2TcpLib**, **m2TcpInit()**, **m2TcpConnEntryGet()**, **m2TcpConnEntrySet()**, **m2TcpDelete()**

---

## m2TcpInit()

**NAME** **m2TcpInit()** – initialize MIB-II TCP-group access

**SYNOPSIS**

```
STATUS m2TcpInit (void)
```

**DESCRIPTION** This routine allocates the resources needed to allow access to the TCP MIB-II variables. This routine must be called before any TCP variables can be accessed.

**RETURNS** OK, always.

**SEE ALSO** **m2TcpLib**, **m2TcpGroupInfoGet()**, **m2TcpConnEntryGet()**, **m2TcpConnEntrySet()**, **m2TcpDelete()**

## m2UdpDelete()

<b>NAME</b>	<b>m2UdpDelete()</b> – delete all resources used to access the UDP group
<b>SYNOPSIS</b>	<b>STATUS</b> m2UdpDelete (void)
<b>DESCRIPTION</b>	This routine frees all the resources allocated at the time the group was initialized. The UDP group should not be accessed after this routine has been called.
<b>RETURNS</b>	OK, always.
<b>SEE ALSO</b>	m2UdpLib, m2UdpInit(), m2UdpGroupInfoGet(), m2UdpTblEntryGet()

---

## m2UdpGroupInfoGet()

<b>NAME</b>	<b>m2UdpGroupInfoGet()</b> – get MIB-II UDP-group scalar variables
<b>SYNOPSIS</b>	<b>STATUS</b> m2UdpGroupInfoGet ( M2_UDP * pUdpInfo            /* pointer to the UDP group structure */ )
<b>DESCRIPTION</b>	This routine fills in the UDP structure at <i>pUdpInfo</i> with the MIB-II UDP scalar variables.
<b>RETURNS</b>	OK, or ERROR if <i>pUdpInfo</i> is not a valid pointer.
<b>ERRNO</b>	S_m2Lib_INVALID_PARAMETER
<b>SEE ALSO</b>	m2UdpLib, m2UdpInit(), m2UdpTblEntryGet(), m2UdpDelete()

---

## m2UdpInit()

<b>NAME</b>	<b>m2UdpInit()</b> – initialize MIB-II UDP-group access
<b>SYNOPSIS</b>	<b>STATUS</b> m2UdpInit (void)
<b>DESCRIPTION</b>	This routine allocates the resources needed to allow access to the UDP MIB-II variables. This routine must be called before any UDP variables can be accessed.
<b>RETURNS</b>	OK, always.
<b>SEE ALSO</b>	m2UdpLib, m2UdpGroupInfoGet(), m2UdpTblEntryGet(), m2UdpDelete()

---

## m2UdpTblEntryGet()

<b>NAME</b>	<b>m2UdpTblEntryGet()</b> – get a UDP MIB-II entry from the UDP list of listeners
<b>SYNOPSIS</b>	<pre> <b>STATUS</b> m2UdpTblEntryGet (     int          search,          /* M2_EXACT_VALUE or M2_NEXT_VALUE */     M2_UDPTBL * pUdpEntry        /* ptr to the requested entry with index */ ) </pre>
<b>DESCRIPTION</b>	This routine traverses the UDP table of listeners and does an M2_EXACT_VALUE or a M2_NEXT_VALUE search based on the <i>search</i> parameter. The calling routine is responsible for supplying a valid MIB-II entry index in the input structure <i>pUdpEntry</i> . The index is made up of the IP address and the local port number. The first entry in the table is retrieved by doing a M2_NEXT_VALUE search with the index fields set to zero.
<b>RETURNS</b>	OK, or ERROR if the input parameter is not specified or a match is not found.
<b>ERRNO</b>	S_m2Lib_INVALID_PARAMETER S_m2Lib_ENTRY_NOT_FOUND
<b>SEE ALSO</b>	m2UdpLib, m2Lib, m2UdpInit(), m2UdpGroupInfoGet(), m2UdpDelete()

## **mach()**

- NAME** **mach()** – return the contents of system register **mach** (also **macl**, **pr**) (SH)
- SYNOPSIS**
- ```
int mach
(
    int taskId          /* task ID, 0 means default task */
)
```
- DESCRIPTION**
- This command extracts the contents of register **mach** from the TCB of a specified task. If *taskId* is omitted or zero, the last task referenced is assumed.
- Similar routines are provided for other system registers (**macl**, **pr**): **macl()**, **pr()**. Note that **pc()** is provided by **usrLib.c**.
- RETURNS**
- The contents of register **mach** (or the requested system register).
- SEE ALSO**
- dbgArchLib**, *VxWorks Programmer's Guide: Debugging*

---

## **malloc()**

- NAME** **malloc()** – allocate a block of memory from the system memory partition (ANSI)
- SYNOPSIS**
- ```
void *malloc
(
    size_t nBytes      /* number of bytes to allocate */
)
```
- DESCRIPTION**
- This routine allocates a block of memory from the free list. The size of the block will be equal to or greater than *nBytes*.
- RETURNS**
- A pointer to the allocated block of memory, or a null pointer if there is an error.
- SEE ALSO**
- memPartLib**, *American National Standard for Information Systems -Programming Language - C, ANSI X3.159-1989: General Utilities (stdlib.h)*

---

## mathHardInit()

**NAME** `mathHardInit()` – initialize hardware floating-point math support

**SYNOPSIS** `void mathHardInit ()`

**DESCRIPTION** This routine places the addresses of the hardware high-level math functions (trigonometric functions, etc.) in a set of global variables. This allows the standard math functions (*e.g.*, `sin()`, `pow()`) to have a single entry point but to be dispatched to the hardware or software support routines, as specified.

This routine is called from `usrConfig.c` if `INCLUDE_HW_FP` is defined. This definition causes the linker to include the floating-point hardware support library.

Certain routines in the floating-point software emulation library do not have equivalent hardware support routines. (These are primarily routines that handle single-precision floating-point numbers.) If no emulation routine address has already been put in the global variable for this function, the address of a dummy routine that logs an error message is placed in the variable; if an emulation routine address is present (the emulation initialization, via `mathSoftInit()`, must be done prior to hardware floating-point initialization), the emulation routine address is left alone. In this way, hardware routines will be used for all available functions, while emulation will be used for the missing functions.

**RETURNS** N/A

**SEE ALSO** `mathHardLib`, `mathSoftInit()`

---

## mathSoftInit()

**NAME** `mathSoftInit()` – initialize software floating-point math support

**SYNOPSIS** `void mathSoftInit (void)`

**DESCRIPTION** This routine places the addresses of the emulated high-level math functions (trigonometric functions, etc.) in a set of global variables. This allows the standard math functions (*e.g.*, `sin()`, `pow()`) to have a single entry point but be dispatched to the hardware or software support routines, as specified.

This routine is called from `usrConfig.c` if `INCLUDE_SW_FP` is defined. This definition causes the linker to include the floating-point emulation library.

***mblen()***

If the system is to use some combination of emulated as well as hardware coprocessor floating points, then this routine should be called before calling **mathHardInit()**.

**RETURNS** N/A

**SEE ALSO** **mathSoftLib**, **mathHardInit()**

**mblen()**

**NAME** **mblen()** – calculate the length of a multibyte character (Unimplemented) (ANSI)

**SYNOPSIS**

```
int mblen
(
    const char * s,
    size_t      n
)
```

**DESCRIPTION** This multibyte character function is unimplemented in VxWorks.

**INCLUDE FILES** **stdlib.h**

**RETURNS** OK, or ERROR if the parameters are invalid.

**SEE ALSO** **ansiStdlib**

**mbstowcs()**

**NAME** **mbstowcs()** – convert a series of multibyte char's to wide char's (Unimplemented) (ANSI)

**SYNOPSIS**

```
size_t mbstowcs
(
    wchar_t *   pwcs,
    const char * s,
    size_t      n
)
```

**DESCRIPTION** This multibyte character function is unimplemented in VxWorks.

**INCLUDE FILES** **stdlib.h**

**RETURNS** OK, or ERROR if the parameters are invalid.

**SEE ALSO** **ansiStdlib**

---

## **mbtowc()**

<b>NAME</b>	<b>mbtowc()</b> – convert a multibyte character to a wide character (Unimplemented) (ANSI)
<b>SYNOPSIS</b>	<pre>int mbtowc (     wchar_t *    pwc,     const char * s,     size_t      n )</pre>
<b>DESCRIPTION</b>	This multibyte character function is unimplemented in VxWorks.
<b>INCLUDE FILES</b>	<code>stdlib.h</code>
<b>RETURNS</b>	OK, or ERROR if the parameters are invalid.
<b>SEE ALSO</b>	<code>ansiStdlib</code>

---

## **mbufShow()**

<b>NAME</b>	<b>mbufShow()</b> – report mbuf statistics
<b>SYNOPSIS</b>	<pre>void mbufShow (void)</pre>
<b>DESCRIPTION</b>	This routine displays the distribution of mbufs in the network.
<b>RETURNS</b>	N/A
<b>SEE ALSO</b>	<code>netShow</code>

## memAddToPool()

<b>NAME</b>	<b>memAddToPool()</b> – add memory to the system memory partition
<b>SYNOPSIS</b>	<pre>void memAddToPool (     char *   pPool,          /* pointer to memory block */     unsigned poolSize      /* block size in bytes */ )</pre>
<b>DESCRIPTION</b>	This routine adds memory to the system memory partition, after the initial allocation of memory to the system memory partition.
<b>RETURNS</b>	N/A
<b>SEE ALSO</b>	<b>memPartLib</b> , <b>memPartAddToPool()</b>

---

## memalign()

<b>NAME</b>	<b>memalign()</b> – allocate aligned memory
<b>SYNOPSIS</b>	<pre>void *memalign (     unsigned alignment,    /* boundary to align to (power of 2) */     unsigned size         /* number of bytes to allocate */ )</pre>
<b>DESCRIPTION</b>	This routine allocates a buffer of size <i>size</i> from the system memory partition. Additionally, it insures that the allocated buffer begins on a memory address evenly divisible by the specified alignment parameter. The alignment parameter must be a power of 2.
<b>RETURNS</b>	A pointer to the newly allocated block, or <b>NULL</b> if the buffer could not be allocated.
<b>SEE ALSO</b>	<b>memLib</b>

---

## memchr()

<b>NAME</b>	<b>memchr()</b> – search a block of memory for a character (ANSI)
<b>SYNOPSIS</b>	<pre>void * memchr (     const void * m,          /* block of memory */     int         c,          /* character to search for */     size_t      n           /* size of memory to search */ )</pre>
<b>DESCRIPTION</b>	This routine searches for the first element of an array of <b>unsigned char</b> , beginning at the address <i>m</i> with size <i>n</i> , that equals <i>c</i> converted to an <b>unsigned char</b> .
<b>INCLUDE FILES</b>	<b>string.h</b>
<b>RETURNS</b>	If successful, it returns the address of the matching element; otherwise, it returns a null pointer.
<b>SEE ALSO</b>	<b>ansiString</b>

---

## memcmp()

<b>NAME</b>	<b>memcmp()</b> – compare two blocks of memory (ANSI)
<b>SYNOPSIS</b>	<pre>int memcmp (     const void * s1,        /* array 1 */     const void * s2,        /* array 2 */     size_t      n           /* size of memory to compare */ )</pre>
<b>DESCRIPTION</b>	This routine compares successive elements from two arrays of <b>unsigned char</b> , beginning at the addresses <i>s1</i> and <i>s2</i> (both of size <i>n</i> ), until it finds elements that are not equal.
<b>INCLUDE FILES</b>	<b>string.h</b>
<b>RETURNS</b>	If all elements are equal, zero. If elements differ and the differing element from <i>s1</i> is greater than the element from <i>s2</i> , the routine returns a positive number; otherwise, it returns a negative number.
<b>SEE ALSO</b>	<b>ansiString</b>

---

## memcpy()

**NAME** `memcpy()` – copy memory from one location to another (ANSI)

**SYNOPSIS**

```
void * memcpy
(
    void *      destination, /* destination of copy */
    const void * source,    /* source of copy */
    size_t      size        /* size of memory to copy */
)
```

**DESCRIPTION** This routine copies *size* characters from the object pointed to by *source* into the object pointed to by *destination*. If copying takes place between objects that overlap, the behavior is undefined.

**INCLUDE FILES** `string.h`

**RETURNS** A pointer to *destination*.

**SEE ALSO** `ansiString`

---

## memDevCreate()

**NAME** `memDevCreate()` – create a memory device

**SYNOPSIS**

```
STATUS memDevCreate
(
    char * name,           /* device name */
    char * base,          /* where to start in memory */
    int   length          /* number of bytes */
)
```

**DESCRIPTION** This routine creates a memory device containing a single file. Memory for the device is simply an absolute memory location beginning at *base*. The *length* parameter indicates the size of memory.

For example, to create the device `/mem/cpu0/`, a device for accessing the entire memory of the local processor, the proper call would be:

```
memDevCreate ("/mem/cpu0/", 0, sysMemTop())
```

The device is created with the specified name, start location, and size.

To open a file descriptor to the memory, use `open()`. Specify a pseudo-file name of the byte offset desired, or open the "raw" file at the beginning and specify a position to seek to. For example, the following call to `open()` allows memory to be read starting at decimal offset 1000.

```
-> fd = open ("/mem/cpu0/1000", O_RDONLY, 0)
```

Pseudo-file name offsets are scanned with "%d".

---

**WARNING:** The `FIOSEEK` operation overrides the offset given via the pseudo-file name at open time.

---

#### EXAMPLE

Consider a system configured with two CPUs in the backplane and a separate dual-ported memory board, each with 1 megabyte of memory. The first CPU is mapped at VMEbus address 0x00400000 (4 Meg.), the second at bus address 0x00800000 (8 Meg.), the dual-ported memory board at 0x00c00000 (12 Meg.). Three devices can be created on each CPU as follows. On processor 0:

```
-> memDevCreate ("/mem/local/", 0, sysMemTop())
...
-> memDevCreate ("/mem/cpu1/", 0x00800000, 0x00100000)
...
-> memDevCreate ("/mem/share/", 0x00c00000, 0x00100000)
```

On processor 1:

```
-> memDevCreate ("/mem/local/", 0, sysMemTop())
...
-> memDevCreate ("/mem/cpu0/", 0x00400000, 0x00100000)
...
-> memDevCreate ("/mem/share/", 0x00c00000, 0x00100000)
```

Processor 0 has a local disk. Data or an object module needs to be passed from processor 0 to processor 1. To accomplish this, processor 0 first calls:

```
-> copy </disk1/module.o >/mem/share/0
```

Processor 1 can then be given the load command:

```
-> ld </mem/share/0
```

#### RETURNS

OK, or `ERROR` if memory is insufficient or the I/O system cannot add the device.

#### ERRNO

`S_ioLib_NO_DRIVER`

#### SEE ALSO

`memDrv`

## memDevCreateDir()

**NAME** memDevCreateDir() – create a memory device for multiple files

**SYNOPSIS**

```
STATUS memDevCreateDir
(
    char *          name,      /* device name */
    MEM_DRV_DIRENTRY * files, /* array of dir. entries - not copied */
    int            numFiles /* number of entries */
)
```

**DESCRIPTION** This routine creates a memory device for a collection of files organized into directories. The given array of directory entry records describes a number of files, some of which may be directories, represented by their own directory entry arrays. The structure may be arbitrarily deep. This effectively allows a file system to be created and installed in VxWorks, for essentially read-only use. The file system structure can be created on the host using the **memdrvbuild** utility.

Note that the array supplied is not copied; a reference to it is kept. This array should not be modified after being passed to **memDevCreateDir()**.

**RETURNS** OK, or ERROR if memory is insufficient or the I/O system cannot add the device.

**ERRNO** S\_ioLib\_NO\_DRIVER

**SEE ALSO** memDrv

---

## memDevDelete()

**NAME** memDevDelete() – delete a memory device

**SYNOPSIS**

```
STATUS memDevDelete
(
    char * name          /* device name */
)
```

**DESCRIPTION** This routine deletes a memory device containing a single file or a collection of files. The device is deleted with its own name.

For example, to delete the device created by **memDevCreate ("/mem/cpu0", 0, sysMemTop())**, the proper call would be:

```
memDevDelete ("/mem/cpu0/");
```

**RETURNS** OK, or **ERROR** if the device doesn't exist.

**SEE ALSO** **memDrv**

---

## **memDrv()**

**NAME** **memDrv()** – install a memory driver

**SYNOPSIS** **STATUS memDrv (void)**

**DESCRIPTION** This routine initializes the memory driver. It must be called first, before any other routine in the driver.

**RETURNS** OK, or **ERROR** if the I/O system cannot install the driver.

**SEE ALSO** **memDrv**

---

## **memFindMax()**

**NAME** **memFindMax()** – find the largest free block in the system memory partition

**SYNOPSIS** **int memFindMax (void)**

**DESCRIPTION** This routine searches for the largest block in the system memory partition free list and returns its size.

**RETURNS** The size, in bytes, of the largest available block.

**SEE ALSO** **memLib, memPartFindMax()**

## memmove()

<b>NAME</b>	<b>memmove()</b> – copy memory from one location to another (ANSI)
<b>SYNOPSIS</b>	<pre>void * memmove (     void *      destination, /* destination of copy */     const void * source,    /* source of copy */     size_t      size        /* size of memory to copy */ )</pre>
<b>DESCRIPTION</b>	This routine copies <i>size</i> characters from the memory location <i>source</i> to the location <i>destination</i> . It ensures that the memory is not corrupted even if <i>source</i> and <i>destination</i> overlap.
<b>INCLUDE FILES</b>	<b>string.h</b>
<b>RETURNS</b>	A pointer to <i>destination</i> .
<b>SEE ALSO</b>	<b>ansiString</b>

---

## memOptionsSet()

<b>NAME</b>	<b>memOptionsSet()</b> – set the debug options for the system memory partition
<b>SYNOPSIS</b>	<pre>void memOptionsSet (     unsigned options /* options for system partition */ )</pre>
<b>DESCRIPTION</b>	<p>This routine sets the debug options for the system memory partition. Two kinds of errors are detected: attempts to allocate more memory than is available, and bad blocks found when memory is freed. In both cases, the following options can be selected for actions to be taken when the error is detected: (1) return the error status, (2) log an error message and return the error status, or (3) log an error message and suspend the calling task.</p> <p>These options are discussed in detail in the library manual entry for <b>memLib</b>.</p>
<b>RETURNS</b>	N/A
<b>SEE ALSO</b>	<b>memLib</b> , <b>memPartOptionsSet()</b>

---

## memPartAddToPool()

**NAME** `memPartAddToPool()` – add memory to a memory partition

**SYNOPSIS**

```
STATUS memPartAddToPool
(
    PART_ID partId,          /* partition to initialize */
    char * pPool,           /* pointer to memory block */
    unsigned poolSize       /* block size in bytes */
)
```

**DESCRIPTION** This routine adds memory to a specified memory partition already created with `memPartCreate()`. The memory added need not be contiguous with memory previously assigned to the partition.

**RETURNS** OK or ERROR.

**ERRNO** `S_smObjLib_NOT_INITIALIZED`, `S_memLib_INVALID_NBYTES`

**SEE ALSO** `memPartLib`, `smMemLib`, `memPartCreate()`

---

## memPartAlignedAlloc()

**NAME** `memPartAlignedAlloc()` – allocate aligned memory from a partition

**SYNOPSIS**

```
void *memPartAlignedAlloc
(
    PART_ID partId,          /* memory partition to allocate from */
    unsigned nBytes,        /* number of bytes to allocate */
    unsigned alignment      /* boundary to align to */
)
```

**DESCRIPTION** This routine allocates a buffer of size *nBytes* from a specified partition. Additionally, it insures that the allocated buffer begins on a memory address evenly divisible by *alignment*. The *alignment* parameter must be a power of 2.

**RETURNS** A pointer to the newly allocated block, or NULL if the buffer could not be allocated.

**SEE ALSO** `memPartLib`

---

## memPartAlloc()

**NAME** memPartAlloc() – allocate a block of memory from a partition

**SYNOPSIS**

```
void *memPartAlloc
(
    PART_ID partId,          /* memory partition to allocate from */
    unsigned nBytes         /* number of bytes to allocate */
)
```

**DESCRIPTION** This routine allocates a block of memory from a specified partition. The size of the block will be equal to or greater than *nBytes*. The partition must already be created with **memPartCreate()**.

**RETURNS** A pointer to a block, or NULL if the call fails.

**ERRNO** S\_smObjLib\_NOT\_INITIALIZED

**SEE ALSO** memPartLib, smMemLib, memPartCreate()

---

## memPartCreate()

**NAME** memPartCreate() – create a memory partition

**SYNOPSIS**

```
PART_ID memPartCreate
(
    char * pPool,           /* pointer to memory area */
    unsigned poolSize      /* size in bytes */
)
```

**DESCRIPTION** This routine creates a new memory partition containing a specified memory pool. It returns a partition ID, which can then be passed to other routines to manage the partition (*i.e.*, to allocate and free memory blocks in the partition). Partitions can be created to manage any number of separate memory pools.

---

**NOTE:** The descriptor for the new partition is allocated out of the system memory partition (*i.e.*, with **malloc()**).

---

**RETURNS** The partition ID, or NULL if there is insufficient memory in the system memory partition for a new partition descriptor.

**SEE ALSO** memPartLib, smMemLib

---

## memPartFindMax()

<b>NAME</b>	<b>memPartFindMax()</b> – find the size of the largest available free block
<b>SYNOPSIS</b>	<pre>int memPartFindMax (     PART_ID partId          /* partition ID */ )</pre>
<b>DESCRIPTION</b>	This routine searches for the largest block in the memory partition free list and returns its size.
<b>RETURNS</b>	The size, in bytes, of the largest available block.
<b>ERRNO</b>	S_smObjLib_NOT_INITIALIZED
<b>SEE ALSO</b>	memLib, smMemLib

---

## memPartFree()

<b>NAME</b>	<b>memPartFree()</b> – free a block of memory in a partition
<b>SYNOPSIS</b>	<pre>STATUS memPartFree (     PART_ID partId,        /* memory partition to add block to */     char * pBlock         /* pointer to block of memory to free */ )</pre>
<b>DESCRIPTION</b>	This routine returns to a partition's free memory list a block of memory previously allocated with <b>memPartAlloc()</b> .
<b>RETURNS</b>	OK, or ERROR if the block is invalid.
<b>ERRNO</b>	S_smObjLib_NOT_INITIALIZED
<b>SEE ALSO</b>	memPartLib, smMemLib, memPartAlloc()

---

## memPartInfoGet()

**NAME** memPartInfoGet() – get partition information

**SYNOPSIS**

```
STATUS memPartInfoGet
(
    PART_ID      partId,    /* partition ID */
    MEM_PART_STATS * ppartStats /* partition stats structure */
)
```

**DESCRIPTION** This routine takes a partition ID and a pointer to a MEM\_PART\_STATS structure. All the parameters of the structure are filled in with the current partition information.

**RETURNS** OK if the structure has valid data, otherwise ERROR.

**SEE ALSO** memShow, memShow()

---

## memPartOptionsSet()

**NAME** memPartOptionsSet() – set the debug options for a memory partition

**SYNOPSIS**

```
STATUS memPartOptionsSet
(
    PART_ID partId,          /* partition to set option for */
    unsigned options        /* memory management options */
)
```

**DESCRIPTION** This routine sets the debug options for a specified memory partition. Two kinds of errors are detected: attempts to allocate more memory than is available, and bad blocks found when memory is freed. In both cases, the error status is returned. There are four error-handling options that can be individually selected:

**MEM\_ALLOC\_ERROR\_LOG\_FLAG**

Log a message when there is an error in allocating memory.

**MEM\_ALLOC\_ERROR\_SUSPEND\_FLAG**

Suspend the task when there is an error in allocating memory (unless the task was spawned with the VX\_UNBREAKABLE option, in which case it cannot be suspended).

**MEM\_BLOCK\_ERROR\_LOG\_FLAG**

Log a message when there is an error in freeing memory.

**MEM\_BLOCK\_ERROR\_SUSPEND\_FLAG**

Suspend the task when there is an error in freeing memory (unless the task was spawned with the **VX\_UNBREAKABLE** option, in which case it cannot be suspended).

These options are discussed in detail in the library manual entry for **memLib**.

**RETURNS** OK or ERROR.  
**ERRNO** S\_smObjLib\_NOT\_INITIALIZED  
**SEE ALSO** memLib, smMemLib

## memPartRealloc()

**NAME** memPartRealloc() – reallocate a block of memory in a specified partition

**SYNOPSIS**

```
void *memPartRealloc
(
    PART_ID partId,          /* partition ID */
    char * pBlock,          /* block to be reallocated */
    unsigned nBytes         /* new block size in bytes */
)
```

**DESCRIPTION** This routine changes the size of a specified block of memory and returns a pointer to the new block. The contents that fit inside the new size (or old size if smaller) remain unchanged. The memory alignment of the new block is not guaranteed to be the same as the original block.

If *pBlock* is NULL, this call is equivalent to **memPartAlloc()**.

**RETURNS** A pointer to the new block of memory, or NULL if the call fails.  
**ERRNO** S\_smObjLib\_NOT\_INITIALIZED  
**SEE ALSO** memLib, smMemLib



---

## memPartShow()

**NAME** `memPartShow()` – show partition blocks and statistics

**SYNOPSIS**

```
STATUS memPartShow
(
    PART_ID partId,          /* partition ID */
    int     type            /* 0 = statistics, 1 = statistics & list */
)
```

**DESCRIPTION** This routine displays statistics about the available and allocated memory in a specified memory partition. It shows the number of bytes, the number of blocks, and the average block size in both free and allocated memory, and also the maximum block size of free memory. It also shows the number of blocks currently allocated and the average allocated block size.

In addition, if *type* is 1, the routine displays a list of all the blocks in the free list of the specified partition.

**RETURNS** OK or ERROR.

**ERRNO** S\_smObjLib\_NOT\_INITIALIZED

**SEE ALSO** `memShow`, `memShow()`, *VxWorks Programmer's Guide: Target Shell*, `windsh`, *Tornado User's Guide: Shell*

---

## memPartSmCreate()

**NAME** `memPartSmCreate()` – create a shared memory partition (VxMP Opt.)

**SYNOPSIS**

```
PART_ID memPartSmCreate
(
    char *   pPool,          /* global address of shared memory area */
    unsigned poolSize       /* size in bytes */
)
```

**DESCRIPTION** This routine creates a shared memory partition that can be used by tasks on all CPUs in the system. It returns a partition ID which can then be passed to generic `memPartLib` routines to manage the partition (*i.e.*, to allocate and free memory blocks in the partition).

*pPool* is the global address of shared memory dedicated to the partition. The memory area pointed to by *pPool* must be in the same address space as the shared memory anchor and shared memory pool.

*poolSize* is the size in bytes of shared memory dedicated to the partition.

Before this routine can be called, the shared memory objects facility must be initialized (see **smMemLib**).

---

**NOTE:** The descriptor for the new partition is allocated out of an internal dedicated shared memory partition. The maximum number of partitions that can be created is **SM\_OBJ\_MAX\_MEM\_PART**.

---

Memory pool size is rounded down to a 16-byte boundary.

<b>AVAILABILITY</b>	This routine is distributed as a component of the unbundled shared memory objects support option, VxMP.
<b>RETURNS</b>	The partition ID, or <b>NULL</b> if there is insufficient memory in the dedicated partition for a new partition descriptor.
<b>ERRNO</b>	<b>S_memLib_NOT_ENOUGH_MEMORY</b> <b>S_smObjLib_LOCK_TIMEOUT</b>
<b>SEE ALSO</b>	<b>smMemLib</b> , <b>memLib</b>

---

## memset()

<b>NAME</b>	<b>memset()</b> – set a block of memory (ANSI)
<b>SYNOPSIS</b>	<pre>void * memset (     void * m,           /* block of memory */     int   c,           /* character to store */     size_t size        /* size of memory */ )</pre>
<b>DESCRIPTION</b>	This routine stores <i>c</i> converted to an <b>unsigned char</b> in each of the elements of the array of <b>unsigned char</b> beginning at <i>m</i> , with size <i>size</i> .
<b>INCLUDE FILES</b>	<b>string.h</b>
<b>RETURNS</b>	A pointer to <i>m</i> .
<b>SEE ALSO</b>	<b>ansiString</b>

---

## memShow()

**NAME** `memShow()` – show system memory partition blocks and statistics

**SYNOPSIS**

```
void memShow
(
    int type          /* 1 = list all blocks in the free list */
)
```

**DESCRIPTION** This routine displays statistics about the available and allocated memory in the system memory partition. It shows the number of bytes, the number of blocks, and the average block size in both free and allocated memory, and also the maximum block size of free memory. It also shows the number of blocks currently allocated and the average allocated block size.

In addition, if *type* is 1, the routine displays a list of all the blocks in the free list of the system partition.

**EXAMPLE**

```
-> memShow 1
FREE LIST:
  num   addr      size
  ---   -
   1   0x3fee18    16
   2   0x3b1434    20
   3   0x4d188   2909400
SUMMARY:
status  bytes     blocks  avg block  max block
-----
current
  free  2909436      3      969812   2909400
  alloc  969060    16102         60         -
cumulative
  alloc 1143340    16365         69         -
```

**RETURNS** N/A

**SEE ALSO** `memShow`, `memPartShow()`, *VxWorks Programmer's Guide: Target Shell*, `windsh`, *Tornado User's Guide: Shell*

---

## memShowInit()

<b>NAME</b>	<b>memShowInit()</b> – initialize the memory partition show facility
<b>SYNOPSIS</b>	<pre>void memShowInit (void)</pre>
<b>DESCRIPTION</b>	<p>This routine links the memory partition show facility into the VxWorks system. These routines are included automatically when this show facility is configured into VxWorks using either of the following methods:</p> <ul style="list-style-type: none"><li>– If you use the configuration header files, define <b>INCLUDE_SHOW_ROUTINES</b> in <b>config.h</b>.</li><li>– If you use the Tornado project facility, select <b>INCLUDE_MEM_SHOW</b>.</li></ul>
<b>RETURNS</b>	N/A
<b>SEE ALSO</b>	<b>memShow</b>

---

## mkdir()

<b>NAME</b>	<b>mkdir()</b> – make a directory
<b>SYNOPSIS</b>	<pre>STATUS mkdir (     const char * dirName    /* directory name */ )</pre>
<b>DESCRIPTION</b>	<p>This command creates a new directory in a hierarchical file system. The <i>dirName</i> string specifies the name to be used for the new directory, and can be either a full or relative pathname.</p> <p>This call is supported by the VxWorks NFS and dosFs file systems.</p>
<b>RETURNS</b>	OK, or ERROR if the directory cannot be created.
<b>SEE ALSO</b>	<b>usrFsLib</b> , <b>rmdir()</b> , <i>VxWorks Programmer's Guide: Target Shell</i>

**mktime()**

---

**mktime()**

<b>NAME</b>	<b>mktime()</b> – convert broken-down time into calendar time (ANSI)
<b>SYNOPSIS</b>	<pre>time_t mktime (     struct tm * timeptr      /* pointer to broken-down structure */ )</pre>
<b>DESCRIPTION</b>	<p>This routine converts the broken-down time, expressed as local time, in the structure pointed to by <i>timeptr</i> into a calendar time value with the same encoding as that of the values returned by the <b>time()</b> function. The original values of the <b>tm_wday</b> and <b>tm_yday</b> components of the <b>tm</b> structure are ignored, and the original values of the other components are not restricted to the ranges indicated in <b>time.h</b>. On successful completion, the values of <b>tm_wday</b> and <b>tm_yday</b> are set appropriately, and the other components are set to represent the specified calendar time, but with their values forced to the ranges indicated in <b>time.h</b>; the final value of <b>tm_mday</b> is not set until <b>tm_mon</b> and <b>tm_year</b> are determined.</p>
<b>INCLUDE FILES</b>	<b>time.h</b>
<b>RETURNS</b>	The calendar time in seconds, or <b>ERROR</b> (-1) if calendar time cannot be calculated.
<b>SEE ALSO</b>	<b>ansiTime</b>

---

**mlock()**

<b>NAME</b>	<b>mlock()</b> – lock specified pages into memory (POSIX)
<b>SYNOPSIS</b>	<pre>int mlock (     const void * addr,     size_t      len )</pre>
<b>DESCRIPTION</b>	<p>This routine guarantees that the specified pages are memory resident. In VxWorks, the <i>addr</i> and <i>len</i> arguments are ignored, since all pages are memory resident.</p>
<b>RETURNS</b>	0 (OK) always.
<b>SEE ALSO</b>	<b>mmanPxLib</b>

---

## mlockall()

<b>NAME</b>	<b>mlockall()</b> – lock all pages used by a process into memory (POSIX)
<b>SYNOPSIS</b>	<pre>int mlockall (     int flags )</pre>
<b>DESCRIPTION</b>	This routine guarantees that all pages used by a process are memory resident. In VxWorks, the <i>flags</i> argument is ignored, since all pages are memory resident.
<b>RETURNS</b>	0 (OK) always.
<b>ERRNO</b>	N/A
<b>SEE ALSO</b>	<b>mmanPxLib</b>

---

## mmuPhysToVirt()

<b>NAME</b>	<b>mmuPhysToVirt()</b> – translate a physical address to a virtual address (ARM)
<b>SYNOPSIS</b>	<pre>void * mmuPhysToVirt (     void * physAddr          /* physical address to be translated */ )</pre>
<b>DESCRIPTION</b>	<p>This function converts a physical address to a virtual address using the information contained within the <b>sysPhysMemDesc</b> structure of the BSP. This routine may be used both by the BSP MMU initialization and by the <b>vm(Base)Lib</b> code.</p> <p>If the BSP has a default mapping where physical and virtual addresses are not identical, then it must provide routines to the cache and MMU architecture code to convert between physical and virtual addresses. If the mapping described within the <b>sysPhysMemDesc</b> structure is accurate, then the BSP may use this routine. If it is not accurate, then routines must be provided within the BSP that are accurate.</p>

---

**NOTE:** This routine simply performs a linear search through the **sysPhysMemDesc** structure looking for the first entry with an address range that includes the given address. Typically, the performance of this should not be a problem, as this routine will generally be called to translate RAM addresses, and by convention, the RAM entries come first in

the structure. If this becomes an issue, the routine could be changed so that a separate structure to **sysPhysMemDesc** is used, containing the information in a more quickly accessible form. In any case, if this is not satisfactory, the BSP can provide its own routines.

---

**SEE ALSO**      **mmuMapLib, mmuVirtToPhys**

**RETURNS**      the virtual address

---

## mmuPro32LibInit()

**NAME** `mmuPro32LibInit()` – initialize module

**SYNOPSIS**

```
STATUS mmuPro32LibInit
(
    int pageSize          /* system pageSize (must be 4KB or 4MB) */
)
```

**DESCRIPTION** Build a dummy translation table that will hold the page table entries for the global translation table. The MMU remains disabled upon completion.

**RETURNS** OK if no error, ERROR otherwise

**ERRNO** S\_mmuLib\_INVALID\_PAGE\_SIZE

**SEE ALSO** `mmuPro32Lib`

---

## mmuSh7700LibInit()

**NAME** `mmuSh7700LibInit()` – initialize module

**SYNOPSIS**

```
STATUS mmuSh7700LibInit
(
    int pageSize
)
```

**DESCRIPTION** Build a dummy translation table that will hold the page table entries for the global translation table. The MMU remains disabled upon completion. Note that this routine is global so that it may be referenced in `usrConfig.c` to pull in the correct `mmuLib` for the specific architecture.

**RETURNS** OK or ERROR

**SEE ALSO** `mmuSh7700Lib`

M

---

## mmuSh7750LibInit()

**NAME** `mmuSh7750LibInit()` – initialize module

**SYNOPSIS**

```
STATUS mmuSh7750LibInit
(
    int pageSize
)
```

**DESCRIPTION** Build a dummy translation table that will hold the page table entries for the global translation table. The MMU remains disabled upon completion. Note that this routine is global so that it may be referenced in `usrConfig.c` to pull in the correct `mmuLib` for the specific architecture.

**RETURNS** OK or ERROR

**SEE ALSO** `mmuSh7750Lib`

---

## mmuVirtToPhys()

**NAME** `mmuVirtToPhys()` – translate a virtual address to a physical address (ARM)

**SYNOPSIS**

```
void * mmuVirtToPhys
(
    void * virtAddr      /* virtual address to be translated */
)
```

**DESCRIPTION** This function converts a virtual address to a physical address using the information contained within the `sysPhysMemDesc` structure of the BSP. This routine may be used both by the BSP MMU initialization and by the `vm(Base)Lib` code.

If the BSP has a default mapping where physical and virtual addresses are not identical, then it must provide routines to the cache and MMU architecture code to convert between physical and virtual addresses. If the mapping described within the `sysPhysMemDesc` structure is accurate, then the BSP may use this routine. If it is not accurate, then routines must be provided within the BSP that are accurate.

---

**NOTE:** This routine simply performs a linear search through the `sysPhysMemDesc` structure looking for the first entry with an address range that includes the given address. Typically, the performance of this should not be a problem, as this routine will generally be called to translate RAM addresses, and by convention, the RAM entries come first in

the structure. If this becomes an issue, the routine could be changed so that a separate structure to **sysPhysMemDesc** is used, containing the information in a more quickly accessible form. In any case, if this is not satisfactory, the BSP can provide its own routines.

---

**SEE ALSO** `mmuMapLib`, `mmuPhysToVirt()`

**RETURNS** the physical address

---

## **modf()**

**NAME** `modf()` – separate a floating-point number into integer and fraction parts (ANSI)

**SYNOPSIS**

```
double modf
(
    double value,          /* value to split */
    double * pIntPart     /* where integer portion is stored */
)
```

**DESCRIPTION** This routine stores the integer portion of *value* in *pIntPart* and returns the fractional portion. Both parts are double precision and will have the same sign as *value*.

**INCLUDE FILES** `math.h`

**RETURNS** The double-precision fractional portion of *value*.

**SEE ALSO** `ansiMath`, `frexp()`, `ldexp()`

---

## **moduleCheck()**

**NAME** `moduleCheck()` – verify checksums on all modules

**SYNOPSIS**

```
STATUS moduleCheck
(
    int options           /* validation options */
)
```

**DESCRIPTION** This routine verifies the checksums on the segments of all loaded modules. If any of the checksums are incorrect, a message is printed to the console, and the routine returns **ERROR**.

By default, only the text segment checksum is validated.

Bits in the *options* parameter may be set to control specific checks:

**MODCHECK\_TEXT**  
Validate the checksum for the TEXT segment (default).

**MODCHECK\_DATA**  
Validate the checksum for the DATA segment.

**MODCHECK\_BSS**  
Validate the checksum for the BSS segment.

**MODCHECK\_NOPRINT**  
Do not print a message (**moduleCheck()** still returns **ERROR** on failure.)

See the definitions in **moduleLib.h**

**RETURNS** **OK**, or **ERROR** if the checksum is invalid.

**SEE ALSO** **moduleLib**

---

## moduleCreate()

**NAME** **moduleCreate()** – create and initialize a module

**SYNOPSIS**

```
MODULE_ID moduleCreate
(
    char * name,           /* module name */
    int   format,         /* object module format */
    int   flags            /* symFlag as passed to loader (see */
                          /* loadModuleAt()) */
)
```

**DESCRIPTION** This routine creates an object module descriptor.

The arguments specify the name of the object module file, the object module format, and an argument specifying which symbols to add to the symbol table. See the **loadModuleAt()** description of *symFlag* for possible *flags* values.

Space for the new module is dynamically allocated.

**RETURNS** **MODULE\_ID**, or **NULL** if there is an error.

SEE ALSO `moduleLib`, `loadModuleAt()`

---

## moduleCreateHookAdd()

**NAME** `moduleCreateHookAdd()` – add a routine to be called when a module is added

**SYNOPSIS**

```
STATUS moduleCreateHookAdd
(
    FUNCPTR moduleCreateHookRtn /* routine called when module is added */
)
```

**DESCRIPTION** This routine adds a specified routine to a list of routines to be called when a module is created. The specified routine should be declared as follows:

```
void moduleCreateHook
(
    MODULE_ID moduleId /* the module ID */
)
```

This routine is called after all fields of the module ID have been filled in.

---

**NOTE:** Modules do not have information about their object segments when they are created. This information is not available until after the entire load process has finished.

---

**RETURNS** OK or ERROR.

SEE ALSO `moduleLib`, `moduleCreateHookDelete()`

---

## moduleCreateHookDelete()

**NAME** `moduleCreateHookDelete()` – delete a previously added module create hook routine

**SYNOPSIS**

```
STATUS moduleCreateHookDelete
(
    FUNCPTR moduleCreateHookRtn /* routine called when module is added */
)
```

**DESCRIPTION** This routine removes a specified routine from the list of routines to be called at each `moduleCreate()` call.

**RETURNS** OK, or ERROR if the routine is not in the table of module create hook routines.

SEE ALSO `moduleLib`, `moduleCreateHookAdd()`

---

## moduleDelete()

**NAME** `moduleDelete()` – delete module ID information (use `unld()` to reclaim space)

**SYNOPSIS**

```
STATUS moduleDelete
(
    MODULE_ID moduleId          /* module to delete */
)
```

**DESCRIPTION** This routine deletes a module descriptor, freeing any space that was allocated for the use of the module ID.

This routine does not free space allocated for the object module itself -- this is done by `unld()`.

**RETURNS** OK or ERROR.

SEE ALSO `moduleLib`

---

## moduleFindByGroup()

**NAME** `moduleFindByGroup()` – find a module by group number

**SYNOPSIS**

```
MODULE_ID moduleFindByGroup
(
    int groupNumber           /* group number to find */
)
```

**DESCRIPTION** This routine searches for a module with a group number matching *groupNumber*.

**RETURNS** `MODULE_ID`, or `NULL` if no match is found.

SEE ALSO `moduleLib`

---

## moduleFindByName()

**NAME** `moduleFindByName()` – find a module by name

**SYNOPSIS**

```
MODULE_ID moduleFindByName
(
    char * moduleName      /* name of module to find */
)
```

**DESCRIPTION** This routine searches for a module with a name matching *moduleName*.

**RETURNS** `MODULE_ID`, or `NULL` if no match is found.

**SEE ALSO** `moduleLib`

---

## moduleFindByNameAndPath()

<b>NAME</b>	<b>moduleFindByNameAndPath()</b> – find a module by file name and path
<b>SYNOPSIS</b>	<pre>MODULE_ID moduleFindByNameAndPath (     char * moduleName,      /* file name to find */     char * pathName        /* path name to find */ )</pre>
<b>DESCRIPTION</b>	This routine searches for a module with a name matching <i>moduleName</i> and path matching <i>pathName</i> .
<b>RETURNS</b>	MODULE_ID, or NULL if no match is found.
<b>SEE ALSO</b>	<b>moduleLib</b>

---

## moduleFlagsGet()

<b>NAME</b>	<b>moduleFlagsGet()</b> – get the flags associated with a module ID
<b>SYNOPSIS</b>	<pre>int moduleFlagsGet (     MODULE_ID moduleId )</pre>
<b>DESCRIPTION</b>	This routine returns the flags associated with a module ID.
<b>RETURNS</b>	The flags associated with the module ID, or NULL if the module ID is invalid.
<b>SEE ALSO</b>	<b>moduleLib</b>

---

## moduleIdListGet()

<b>NAME</b>	<b>moduleIdListGet()</b> – get a list of loaded modules
<b>SYNOPSIS</b>	<pre>int moduleIdListGet (     MODULE_ID * idList,      /* array of module IDs to be filled in */     int        maxModules   /* max modules idList can accommodate */ )</pre>
<b>DESCRIPTION</b>	This routine provides the calling task with a list of all loaded object modules. An unsorted list of module IDs for no more than <i>maxModules</i> modules is put into <i>idList</i> .
<b>RETURNS</b>	The number of modules put into the ID list, or <b>ERROR</b> .
<b>SEE ALSO</b>	<b>moduleLib</b>

---

## moduleInfoGet()

<b>NAME</b>	<b>moduleInfoGet()</b> – get information about an object module
<b>SYNOPSIS</b>	<pre>STATUS moduleInfoGet (     MODULE_ID    moduleId,  /* module to return information about */     MODULE_INFO * pModuleInfo /* pointer to module info struct */ )</pre>
<b>DESCRIPTION</b>	This routine fills in a <b>MODULE_INFO</b> structure with information about the specified module.
<b>RETURNS</b>	OK or <b>ERROR</b> .
<b>SEE ALSO</b>	<b>moduleLib</b>

## **moduleNameGet()**

**NAME** `moduleNameGet()` – get the name associated with a module ID

**SYNOPSIS**

```
char * moduleNameGet
(
    MODULE_ID moduleId
)
```

**DESCRIPTION** This routine returns a pointer to the name associated with a module ID.

**RETURNS** A pointer to the module name, or `NULL` if the module ID is invalid.

**SEE ALSO** `moduleLib`

---

## **moduleSegFirst()**

**NAME** `moduleSegFirst()` – find the first segment in a module

**SYNOPSIS**

```
SEGMENT_ID moduleSegFirst
(
    MODULE_ID moduleId      /* module to get segment from */
)
```

**DESCRIPTION** This routine returns information about the first segment of a module descriptor.

**RETURNS** A pointer to the segment ID, or `NULL` if the segment list is empty.

**SEE ALSO** `moduleLib`, `moduleSegGet()`

---

## moduleSegGet()

**NAME** `moduleSegGet()` – get (delete and return) the first segment from a module

**SYNOPSIS**

```
SEGMENT_ID moduleSegGet
(
    MODULE_ID moduleId      /* module to get segment from */
)
```

**DESCRIPTION** This routine returns information about the first segment of a module descriptor, and then deletes the segment from the module.

**RETURNS** A pointer to the segment ID, or NULL if the segment list is empty.

**SEE ALSO** `moduleLib`, `moduleSegFirst()`

---

## moduleSegNext()

**NAME** `moduleSegNext()` – find the next segment in a module

**SYNOPSIS**

```
SEGMENT_ID moduleSegNext
(
    SEGMENT_ID segmentId    /* segment whose successor is to be found */
)
```

**DESCRIPTION** This routine returns the segment in the list immediately following *segmentId*.

**RETURNS** A pointer to the segment ID, or NULL if there is no next segment.

**SEE ALSO** `moduleLib`



## moduleShow()

**NAME** `moduleShow()` – show the current status for all the loaded modules

**SYNOPSIS**

```
STATUS moduleShow
(
    char * moduleNameOrId,    /* name or ID of the module to show */
    int   options             /* display options */
)
```

**DESCRIPTION** This routine displays a list of the currently loaded modules and some information about where the modules are loaded.

The specific information displayed depends on the format of the object modules. In the case of a.out and ECOFF object modules, `moduleShow()` displays the start of the text, data, and BSS segments.

If `moduleShow()` is called with no arguments, a summary list of all loaded modules is displayed. It can also be called with an argument, `moduleNameOrId`, which can be either the name of a loaded module or a module ID. If it is called with either of these, more information about the specified module will be displayed.

**RETURNS** OK or ERROR.

**SEE ALSO** `moduleLib`, *VxWorks Programmer's Guide: Target Shell*, `windsh`, *Tornado User's Guide: Shell*

---

## mountdInit()

**NAME** `mountdInit()` – initialize the mount daemon

**SYNOPSIS**

```
STATUS mountdInit
(
    int     priority,          /* priority of the mount daemon */
    int     stackSize,        /* stack size of the mount daemon */
    FUNCPTR authHook,         /* hook to run to authorize each request */
    int     nExports,         /* maximum number of exported file systems */
    int     options           /* currently unused - set to 0 */
)
```

**DESCRIPTION** This routine spawns a mount daemon if one does not already exist. Defaults for the `priority` and `stackSize` arguments are in the global variables `mountdPriorityDefault` and

**mountdStackSizeDefault**, and are initially set to **MOUNTD\_PRIORITY\_DEFAULT** and **MOUNTD\_STACKSIZE\_DEFAULT** respectively.

Normally, no authorization checking is performed by either **mountd** or **nfsd**. To add authorization checking, set **authHook** to point to a routine declared as follows:

```
nfsstat routine
(
    int          progNum,      /* RPC program number */
    int          versNum,     /* RPC program version number */
    int          procNum,     /* RPC procedure number */
    struct sockaddr_in clientAddr, /* address of the client */
    MOUNTD_ARGUMENT * mountdArg /* argument of the call */
)
```

The **authHook** callback must return **OK** if the request is authorized, and any defined NFS error code (usually **NFSERR\_ACCES**) if not.

#### VXWORKS AE PROTECTION DOMAINS

Under VxWorks AE, you can call **mountdInit()** from within the kernel protection domain only, and the function referenced in the **authHook** parameter must reside in the kernel protection domain. This restriction does not apply under non-AE versions of VxWorks.

**RETURNS** OK, or **ERROR** if the mount daemon could not be correctly initialized.

**SEE ALSO** **mountLib**

---

## mqPxLibInit()

**NAME** **mqPxLibInit()** – initialize the POSIX message queue library

**SYNOPSIS**

```
int mqPxLibInit
(
    int hashSize          /* log2 of number of hash buckets */
)
```

**DESCRIPTION** This routine initializes the POSIX message queue facility. If **hashSize** is 0, the default value is taken from **MQ\_HASH\_SIZE\_DEFAULT**.

**RETURNS** **OK** or **ERROR**.

**SEE ALSO** **mqPxLib**

## mqPxShowInit()

- NAME** `mqPxShowInit()` – initialize the POSIX message queue show facility
- SYNOPSIS** `STATUS mqPxShowInit (void)`
- DESCRIPTION** This routine links the POSIX message queue show routine into the VxWorks system. It is called automatically when this show facility is configured into VxWorks using either of the following methods:
- If you use the configuration header files, define `INCLUDE_SHOW_ROUTINES` in `config.h`.
  - If you use the Tornado project facility, select `INCLUDE_POSIX_MQ_SHOW`.
- RETURNS** `OK`, or `ERROR` if an error occurs installing the file pointer `show` routine.
- SEE ALSO** `mqPxShow`
- 

## mq\_close()

- NAME** `mq_close()` – close a message queue (POSIX)
- SYNOPSIS**

```
int mq_close
(
    mqd_t mqdes          /* message queue descriptor */
)
```
- DESCRIPTION** This routine is used to indicate that the calling task is finished with the specified message queue `mqdes`. The `mq_close()` call deallocates any system resources allocated by the system for use by this task for its message queue. The behavior of a task that is blocked on either a `mq_send()` or `mq_receive()` is undefined when `mq_close()` is called. The `mqdes` parameter will no longer be a valid message queue ID.
- RETURNS** 0 (`OK`) if the message queue is closed successfully, otherwise -1 (`ERROR`).
- ERRNO** `EBADF`
- SEE ALSO** `mqPxLib`, `mq_open()`

---

## mq\_getattr()

**NAME** `mq_getattr()` – get message queue attributes (POSIX)

**SYNOPSIS**

```
int mq_getattr
(
    mqd_t          mqdes, /* message queue descriptor */
    struct mq_attr * pMqStat /* buffer in which to return attributes */
)
```

**DESCRIPTION** This routine gets status information and attributes associated with a specified message queue *mqdes*. Upon return, the following members of the `mq_attr` structure referenced by *pMqStat* will contain the values set when the message queue was created but with modifications made by subsequent calls to `mq_setattr()`:

`mq_flags`  
May be modified by `mq_setattr()`.

The following were set at message queue creation:

`mq_maxmsg`  
Maximum number of messages.

`mq_msgsize`  
Maximum message size.

`mq_curmsgs`  
The number of messages currently in the queue.

**RETURNS** 0 (OK) if message attributes can be determined, otherwise -1 (ERROR).

**ERRNO** EBADF

**SEE ALSO** `mqPxBLib`, `mq_open()`, `mq_send()`, `mq_setattr()`

---

## mq\_notify()

**NAME** `mq_notify()` – notify a task that a message is available on a queue (POSIX)

**SYNOPSIS**

```
int mq_notify
(
    mqd_t          mqdes,          /* message queue descriptor */
    const struct sigevent * pNotification /* real-time signal */
)
```

**DESCRIPTION**

If *pNotification* is not NULL, this routine attaches the specified *pNotification* request by the calling task to the specified message queue *mqdes* associated with the calling task. The real-time signal specified by *pNotification* will be sent to the task when the message queue changes from empty to non-empty. If a task has already attached a notification request to the message queue, all subsequent attempts to attach a notification to the message queue will fail. A task is able to attach a single notification to each *mqdes* it has unless another task has already attached one.

If *pNotification* is NULL and the task has previously attached a notification request to the message queue, the attached notification request is detached and the queue is available for another task to attach a notification request.

If a notification request is attached to a message queue and any task is blocked in `mq_receive()` waiting to receive a message when a message arrives at the queue, then the appropriate `mq_receive()` will be completed and the notification request remains pending.

**RETURNS** 0 (OK) if successful, otherwise -1 (ERROR).

**ERRNO** EBADF, EBUSY, EINVAL

**SEE ALSO** `mqPxBLib`, `mq_open()`, `mq_send()`

---

## mq\_open()

**NAME** `mq_open()` – open a message queue (POSIX)

**SYNOPSIS**

```
mqd_t mq_open
(
    const char * mqName,      /* name of queue to open */
    int         oflags,       /* open flags */
    ...             /* extra optional parameters */
)
```

**DESCRIPTION** This routine establishes a connection between a named message queue and the calling task. After a call to `mq_open()`, the task can reference the message queue using the address returned by the call. The message queue remains usable until the queue is closed by a successful call to `mq_close()`.

The *oflags* argument controls whether the message queue is created or merely accessed by the `mq_open()` call. The following flag bits can be set in *oflags*:

### **O\_RDONLY**

Open the message queue for receiving messages. The task can use the returned message queue descriptor with `mq_receive()`, but not `mq_send()`.

### **O\_WRONLY**

Open the message queue for sending messages. The task can use the returned message queue descriptor with `mq_send()`, but not `mq_receive()`.

### **O\_RDWR**

Open the queue for both receiving and sending messages. The task can use any of the functions allowed for **O\_RDONLY** and **O\_WRONLY**.

Any combination of the remaining flags can be specified in *oflags*:

### **O\_CREAT**

This flag is used to create a message queue if it does not already exist. If **O\_CREAT** is set and the message queue already exists, then **O\_CREAT** has no effect except as noted below under **O\_EXCL**. Otherwise, `mq_open()` creates a message queue. The **O\_CREAT** flag requires a third and fourth argument: *mode*, which is of type `mode_t`, and *pAttr*, which is of type pointer to an `mq_attr` structure. The value of *mode* has no effect in this implementation. If *pAttr* is `NULL`, the message queue is created with implementation-defined default message queue attributes. If *pAttr* is non-`NULL`, the message queue attributes `mq_maxmsg` and `mq_msgsize` are set to the values of the corresponding members in the `mq_attr` structure referred to by *pAttr*; if either attribute is less than or equal to zero, an error is returned and `errno` is set to `EINVAL`.

### **O\_EXCL**

This flag is used to test whether a message queue already exists. If **O\_EXCL** and **O\_CREAT** are set, `mq_open()` fails if the message queue name exists.

#### O\_NONBLOCK

The setting of this flag is associated with the open message queue descriptor and determines whether a **mq\_send()** or **mq\_receive()** will wait for resources or messages that are not currently available, or fail with **errno** set to **EAGAIN**.

The **mq\_open()** call does not add or remove messages from the queue.

---

**NOTE:** Some POSIX functionality is not yet supported:

- A message queue cannot be closed with calls to **\_exit()** or **exec()**.
  - A message queue cannot be implemented as a file.
  - Message queue names will not appear in the file system.
- 

#### RETURNS

A message queue descriptor, otherwise -1 (**ERROR**).

#### ERRNO

EEXIST, EINVAL, ENOENT, ENOSPC

#### SEE ALSO

**mqPxBLib**, **mq\_send()**, **mq\_receive()**, **mq\_close()**, **mq\_setattr()**, **mq\_getattr()**, **mq\_unlink()**

---

## mq\_receive()

#### NAME

**mq\_receive()** – receive a message from a message queue (POSIX)

#### SYNOPSIS

```
ssize_t mq_receive
(
    mqd_t mqdes,           /* message queue descriptor */
    void * pMsg,          /* buffer to receive message */
    size_t msgLen,        /* size of buffer, in bytes */
    int * pMsgPrio        /* if not NULL, priority of message */
)
```

#### DESCRIPTION

This routine receives the oldest of the highest priority message from the message queue specified by *mqdes*. If the size of the buffer in bytes, specified by the *msgLen* argument, is less than the **mq\_msgsize** attribute of the message queue, **mq\_receive()** will fail and return an error. Otherwise, the selected message is removed from the queue and copied to *pMsg*.

If *pMsgPrio* is not **NULL**, the priority of the selected message will be stored in *pMsgPrio*.

If the message queue is empty and **O\_NONBLOCK** is not set in the message queue's description, **mq\_receive()** will block until a message is added to the message queue, or until it is interrupted by a signal. If more than one task is waiting to receive a message when a message arrives at an empty queue, the task of highest priority that has been

waiting the longest will be selected to receive the message. If the specified message queue is empty and `O_NONBLOCK` is set in the message queue's description, no message is removed from the queue, and `mq_receive()` returns an error.

<b>RETURNS</b>	The length of the selected message in bytes, otherwise -1 ( <b>ERROR</b> ).
<b>ERRNO</b>	<code>EAGAIN</code> , <code>EBADF</code> , <code>EMSGSIZE</code> , <code>EINTR</code>
<b>SEE ALSO</b>	<code>mqPxLib</code> , <code>mq_send()</code>

---

## mq\_send()

**NAME** `mq_send()` – send a message to a message queue (POSIX)

**SYNOPSIS**

```
int mq_send
(
    mqd_t      mqdes,          /* message queue descriptor */
    const void * pMsg,        /* message to send */
    size_t     msgLen,        /* size of message, in bytes */
    int        msgPrio        /* priority of message */
)
```

**DESCRIPTION** This routine adds the message *pMsg* to the message queue *mqdes*. The *msgLen* parameter specifies the length of the message in bytes pointed to by *pMsg*. The value of *pMsg* must be less than or equal to the `mq_msgsize` attribute of the message queue, or `mq_send()` will fail.

If the message queue is not full, `mq_send()` will behave as if the message is inserted into the message queue at the position indicated by the *msgPrio* argument. A message with a higher numeric value for *msgPrio* is inserted before messages with a lower value. The value of *msgPrio* must be less than or equal to 31.

If the specified message queue is full and `O_NONBLOCK` is not set in the message queue's, `mq_send()` will block until space becomes available to queue the message, or until it is interrupted by a signal. The priority scheduling option is supported in the event that there is more than one task waiting on space becoming available. If the message queue is full and `O_NONBLOCK` is set in the message queue's description, the message is not queued, and `mq_send()` returns an error.

### USE BY INTERRUPT SERVICE ROUTINES

This routine can be called by interrupt service routines as well as by tasks. This is one of the primary means of communication between an interrupt service routine and a task. If `mq_send()` is called from an interrupt service routine, it will behave as if the

O\_NONBLOCK flag were set.

- RETURNS** 0 (OK), otherwise -1 (ERROR).
- ERRNO** EAGAIN, EBADF, EINTR, EINVAL, EMSGSIZE
- SEE ALSO** mqPxLib, mq\_receive()

---

## mq\_setattr()

**NAME** mq\_setattr() – set message queue attributes (POSIX)

**SYNOPSIS**

```
int mq_setattr
(
    mqd_t          mqdes,      /* message queue descriptor */
    const struct mq_attr * pMqStat, /* new attributes */
    struct mq_attr *   pOldMqStat /* old attributes */
)
```

**DESCRIPTION** This routine sets attributes associated with the specified message queue *mqdes*. The message queue attributes corresponding to the following members defined in the **mq\_attr** structure are set to the specified values upon successful completion of the call:

mq\_flags

The value the O\_NONBLOCK flag.

If *pOldMqStat* is non-NULL, **mq\_setattr()** will store, in the location referenced by *pOldMqStat*, the previous message queue attributes and the current queue status. These values are the same as would be returned by a call to **mq\_getattr()** at that point.

- RETURNS** 0 (OK) if attributes are set successfully, otherwise -1 (ERROR).
- ERRNO** EBADF
- SEE ALSO** mqPxLib, mq\_open(), mq\_send(), mq\_getattr()

---

## mq\_unlink()

<b>NAME</b>	<b>mq_unlink()</b> – remove a message queue (POSIX)
<b>SYNOPSIS</b>	<pre>int mq_unlink (     const char * mqName      /* name of message queue */ )</pre>
<b>DESCRIPTION</b>	This routine removes the message queue named by the pathname <i>mqName</i> . After a successful call to <b>mq_unlink()</b> , a call to <b>mq_open()</b> on the same message queue will fail if the flag <b>O_CREAT</b> is not set. If one or more tasks have the message queue open when <b>mq_unlink()</b> is called, removal of the message queue is postponed until all references to the message queue have been closed.
<b>RETURNS</b>	0 (OK) if the message queue is unlinked successfully, otherwise -1 (ERROR).
<b>ERRNO</b>	ENOENT
<b>SEE ALSO</b>	mqPxLib, mq_close(), mq_open()

---

## mRegs()

<b>NAME</b>	<b>mRegs()</b> – modify registers
<b>SYNOPSIS</b>	<pre>STATUS mRegs (     char * regName,          /* register name, NULL for all */     int   taskNameOrId     /* task name or task ID, 0 = default task */ )</pre>
<b>DESCRIPTION</b>	This command modifies the specified register for the specified task. If <i>taskNameOrId</i> is omitted or zero, the last task referenced is assumed. If the specified register is not found, it prints out the valid register list and returns <b>ERROR</b> . If no register is specified, it sequentially prompts the user for new values for a task's registers. It displays each register and the current contents of that register, in turn. The user can respond in one of several ways:
<b>RETURN</b>	Do not change this register, but continue, prompting at the next register.

**mRouteAdd()**

number

Set this register to *number*.

. (dot)

Do not change this register, and quit.

EOF

Do not change this register, and quit.

All numbers are entered and displayed in hexadecimal, except floating-point values, which may be entered in double precision.

**RETURNS** OK, or **ERROR** if the task or register does not exist.

**SEE ALSO** **usrLib, m()**, *VxWorks Programmer's Guide: Target Shell*, **windsh**, *Tornado User's Guide: Shell*

---

## mRouteAdd()

**NAME** **mRouteAdd()** – add multiple routes to the same destination

**SYNOPSIS**

```

STATUS mRouteAdd
(
    char * pDest,           /* destination addr in internet dot notation */
    char * pGate,          /* gateway address in internet dot notation */
    long  mask,            /* mask for destination */
    int   tos,             /* type of service */
    int   flags             /* route flags */
)

```

**DESCRIPTION** This routine is similar to **routeAdd()**, except that you can use multiple **mRouteAdd()** calls to add multiple routes to the same location. Use *pDest* to specify the destination, *pGate* to specify the gateway to that destination, *mask* to specify destination mask, and *tos* to specify the type of service. For *tos*, **netinet/ip.h** defines the following constants as valid values:

```

IPTOS_LOWDELAY
IPTOS_THROUGHPUT
IPTOS_RELIABILITY
IPTOS_MINCOST

```

Use *flags* to specify any flags you want to associate with this entry. The valid non-zero values are **RTF\_HOST** and **RTF\_CLONING** defined in **net/route.h**.

**EXAMPLE** To add a route to the 90.0.0.0 network through 91.0.0.3:

```
-> mRouteAdd ("90.0.0.0", "91.0.0.3", 0xffffffff00, 0, 0);
```

Using **mRouteAdd()**, you could create multiple routes to the same destination. VxWorks would distinguish among these routes based on factors such as the netmask or the type of service. Thus, it is perfectly legal to say:

```
-> mRouteAdd ("90.0.0.0", "91.0.0.3", 0xffffffff00, 0, 0);
-> mRouteAdd ("90.0.0.0", "91.0.0.254", 0xffff0000, 0, 0);
```

This adds two routes to the same network, "90.0.0.0", that go by two different gateways. The differentiating factor is the netmask.

This routine adds a route of type **M2\_ipRouteProto\_other**, which is a static route. This route will not be modified or deleted until a call to **mRouteDelete()** removes it.

**RETURNS** OK or ERROR.

**SEE ALSO** **routeLib**, **mRouteEntryAdd()**, **mRouteDelete()**, **routeAdd()**

---

## mRouteDelete()

M

**NAME** **mRouteDelete()** – delete a route from the routing table

**SYNOPSIS** **STATUS mRouteDelete**

```
(
char * pDest,          /* destination address */
long  mask,           /* mask for destination */
int   tos,            /* type of service */
int   flags           /* either 0 or RTF_HOST */
)
```

**DESCRIPTION** This routine deletes a routing table entry as specified by the destination, *pDest*, the destination mask, *mask*, and type of service, *tos*. The *tos* values are as defined in the reference entry for **mRouteAdd()**.

**EXAMPLE** Consider the case of a route added in the following manner:

```
-> mRouteAdd ("90.0.0.0", "91.0.0.3", 0xffffffff00, 0, 0);
```

To delete a route that was added in the above manner, call **mRouteDelete()** as follows:

```
-> mRouteDelete("90.0.0.0", 0xffffffff00, 0);
```

If the netmask and or type of service do not match, the route is not deleted.

The value of *flags* should be `RTF_HOST` for host routes, `RTF_CLONING` for routes which need to be cloned, and 0 in all other cases.

**RETURNS** OK or ERROR.

**SEE ALSO** routeLib, mRouteAdd()

---

## mRouteEntryAdd()

**NAME** mRouteEntryAdd() – add a protocol-specific route to the routing table

**SYNOPSIS**

```
STATUS mRouteEntryAdd
(
    long destIp,           /* destination address, network order */
    long gateIp,          /* gateway address, network order */
    long mask,            /* mask for destination, network order */
    int  tos,             /* type of service */
    int  flags,           /* route flags */
    int  proto            /* routing protocol */
)
```

**DESCRIPTION** For a single destination *destIp*, this routine can add additional routes *gateIp* to the routing table. The different routes are distinguished by the gateway, a destination mask *mask*, the type of service *tos* and associated flag values *flags*. Valid values for *flags* are 0, `RTF_HOST`, `RTF_CLONING` (defined in `net/route.h`). The *proto* parameter identifies the protocol that generated this route. Values for *proto* may be found in `m2Lib.h`. The *tos* parameter takes one of following values (defined in `netinet/ip.h`):

```
    IPTOS_LOWDELAY
    IPTOS_THROUGHPUT
    IPTOS_RELIABILITY
    IPTOS_MINCOST
```

**RETURNS** OK or ERROR.

**SEE ALSO** routeLib, m2Lib.h, mRouteAdd(), mRouteDelete()

---

## mRouteEntryDelete()

**NAME** `mRouteEntryDelete()` – delete route from the routing table

**SYNOPSIS**

```
STATUS mRouteEntryDelete
(
    long destIp,          /* destination address, network order */
    long gateIp,         /* gateway address, network order */
    long mask,           /* mask for destination, network order */
    int  tos,            /* type of service */
    int  flags,          /* route flags */
    int  proto           /* routing protocol */
)
```

**DESCRIPTION** This routine deletes a protocol-specific route from the routing table. Specify the route using a destination *pDest*, a gateway *pGate*, a destination mask *mask*, the type of service *tos*, a *flags* value, and a *proto* value that identifies the routing protocol that added the route. The valid values for *flags* are 0 and `RTF_HOST` (defined in `net/route.h`). Values for *proto* may be found in `m2Lib.h` and *tos* is one of the following values defined in `netinet/ip.h`:

```
    IPTOS_LOWDELA
    IPTOS_THROUGHPU
    IPTOS_RELIABILIT
    IPTOS_MINCOST
```

An existing route is deleted only if it is owned by the protocol specified by *proto*.

**RETURNS** `OK` or `ERROR`.

**SEE ALSO** `routeLib`

---

## mRouteShow()

**NAME** `mRouteShow()` – display all IP routes (verbose information)

**SYNOPSIS**

```
void mRouteShow (void)
```

**DESCRIPTION** This routine displays the list of destinations in the routing table along with the next-hop gateway and associated interface. It also displays the netmask for a route (to handle classless routes which use arbitrary values for that field) and the value which indicates the route's creator, as well as any type-of-service information.

When multiple routes exist to the same destination with the same netmask, the IP forwarding process only uses the first route entry with the lowest administrative weight. The remaining entries (listed as additional routes) use the same address and netmask. One of those entries will replace the primary route if it is deleted.

Some configuration is required when this routine is to be used remotely over the network, *e.g.*, through a **telnet** session or through the host shell using **WDB\_COMM\_NETWORK**. If more than 5 routes are expected in the table the parameter **RT\_BUFFERED\_DISPLAY** should be set to **TRUE** to prevent a possible deadlock. This requires a buffer whose size can be set with **RT\_DISPLAY\_MEMORY**. It will limit the number of routes that can be displayed (each route requires approx. 90 bytes).

**RETURNS** N/A

**SEE ALSO** netShow

---

## msgQCreate()

**NAME** msgQCreate() – create and initialize a message queue

**SYNOPSIS** MSG\_Q\_ID msgQCreate

```
(  
    int maxMsgs,           /* max messages that can be queued */  
    int maxMsgLength,     /* max bytes in a message */  
    int options           /* message queue options */  
)
```

**DESCRIPTION** This routine creates a message queue capable of holding up to *maxMsgs* messages, each up to *maxMsgLength* bytes long. The routine returns a message queue ID used to identify the created message queue in all subsequent calls to routines in this library. The queue can be created with the following options:

**MSG\_Q\_FIFO** (0x00)  
queue pending tasks in FIFO order.

**MSG\_Q\_PRIORITY** (0x01)  
queue pending tasks in priority order.

**MSG\_Q\_EVENTSEND\_ERR\_NOTIFY** (0x02)  
When a message is sent, if a task is registered for events and the actual sending of events fails, a value of **ERROR** is returned and the **errno** is set accordingly. This option is off by default.

**RETURNS** MSG\_Q\_ID, or NULL if error.

**ERRNO** S\_memLib\_NOT\_ENOUGH\_MEMORY, S\_intLib\_NOT\_ISR\_CALLABLE

**SEE ALSO** msgQLib, msgQSmLib

---

## msgQDelete()

<b>NAME</b>	<code>msgQDelete()</code> – delete a message queue
<b>SYNOPSIS</b>	<pre>STATUS msgQDelete (     MSG_Q_ID msgQId          /* message queue to delete */ )</pre>
<b>DESCRIPTION</b>	This routine deletes a message queue. All tasks pending on either <code>msgQSend()</code> , <code>msgQReceive()</code> or pending for the reception of events meant to be sent from the message queue will unblock and return <code>ERROR</code> . When this function returns, <i>msgQId</i> is no longer a valid message queue ID.
<b>RETURNS</b>	<code>OK</code> on success or <code>ERROR</code> otherwise.
<b>ERRNO</b>	<code>S_objLib_OBJ_ID_ERROR</code> Message queue ID is invalid <code>S_intLib_NOT_ISR_CALLABLE</code> Routine cannot be called from ISR <code>S_distLib_NO_OBJECT_DESTROY</code> Deleting a distributed message queue is not permitted <code>S_smObjLib_NO_OBJECT_DESTROY</code> Deleting a shared message queue is not permitted
<b>SEE ALSO</b>	<code>msgQLib</code> , <code>msgQSmLib</code>

---

## msgQDistCreate()

<b>NAME</b>	<code>msgQDistCreate()</code> – create a distributed message queue (VxFusion Opt.)
<b>SYNOPSIS</b>	<pre>MSG_Q_ID msgQDistCreate (     int maxMsgs,             /* max messages that can be queued */     int maxMsgLength,       /* max bytes in a message */     int options              /* message queue options */ )</pre>

<b>DESCRIPTION</b>	<p>This routine creates a distributed message queue capable of holding up to <i>maxMsgs</i> messages, each up to <i>maxMsgLength</i> bytes long. This routine returns a message queue ID used to identify the created message queue. The queue can be created with the following options:</p> <p><b>MSG_Q_FIFO</b> (0x00) The queue pends tasks in FIFO order.</p> <p><b>MSG_Q_PRIORITY</b> (0x01) The queue pends tasks in priority order. Remote tasks share the same priority level.</p> <p>The global message queue identifier returned can be used directly by generic message queue handling routines in <b>msgQLib</b>, such as, <b>msgQSend()</b>, <b>msgQReceive()</b>, and <b>msgQNumMsgs()</b>.</p>
<b>AVAILABILITY</b>	<p>This routine is distributed as a component of the unbundled distributed message queues option, VxFusion.</p>
<b>RETURNS</b>	<p><b>MSG_Q_ID</b>, or NULL if there is an error.</p>
<b>ERRNO</b>	<p><b>S_memLib_NOT_ENOUGH_MEMORY</b> If the routine is unable to allocate memory for message queues and message buffers.</p> <p><b>S_intLib_NOT_ISR_CALLABLE</b> If the routine is called from an interrupt service routine.</p> <p><b>S_msgQLib_INVALID_QUEUE_TYPE</b> If the type of queue is invalid.</p> <p><b>S_msgQDistLib_INVALID_MSG_LENGTH</b> If the message is too long for the VxFusion network layer.</p>
<b>SEE ALSO</b>	<p><b>msgQDistLib</b>, <b>msgQLib</b></p>

---

## msgQDistGrpAdd()

<b>NAME</b>	<code>msgQDistGrpAdd()</code> – add a distributed message queue to a group (VxFusion Opt.)
<b>SYNOPSIS</b>	<pre>MSG_Q_ID msgQDistGrpAdd (     char *      distGrpName, /* new or existing group name */     MSG_Q_ID   msgQId,      /* message queue to add to the group */     DIST_GRP_OPT options    /* group message queue options - UNUSED */ )</pre>
<b>DESCRIPTION</b>	<p>This routine adds the queue identified by the argument <i>msgQId</i> to a group with the ASCII name specified by the argument <i>distGrpName</i>.</p> <p>Multicasting is based on distributed message queue groups. If the group does not exist, one is created. Any number of message queues from different nodes can be bound to a single group. In addition, a message queue can be added into any number of groups; <code>msgQDistGrpAdd()</code> must be called for each group of which the message queue is to be a member.</p> <p>The <i>options</i> parameter is presently unused and must be set to 0.</p> <p>This routine returns a message queue ID, <code>MSG_Q_ID</code>, that can be used directly by <code>msgQDistSend()</code> or by the generic <code>msgQSend()</code> routine. Do not call the <code>msgQReceive()</code> or <code>msgQNumMsgs()</code> routines or their distributed counterparts, <code>msgQDistReceive()</code> and <code>msgQDistNumMsgs()</code>, with a group message queue ID.</p> <p>As with <code>msgQDistCreate()</code>, use <code>distNameAdd()</code> to add the group message queue ID returned by this routine to the distributed name database so that the ID can be used by tasks on other nodes.</p>
<b>AVAILABILITY</b>	This routine is distributed as a component of the unbundled distributed message queues option, VxFusion.
<b>RETURNS</b>	<code>MSG_Q_ID</code> , or NULL if there is an error.
<b>ERRNO</b>	<code>S_msgQDistGrpLib_NAME_TOO_LONG</code> The name of the group is too long.
	<code>S_msgQDistGrpLib_INVALID_OPTION</code> The <i>options</i> parameter is invalid.
	<code>S_msgQDistGrpLib_DATABASE_FULL</code> The group database is full.
	<code>S_distLib_OBJ_ID_ERROR</code> The <i>msgQId</i> parameter is not a distributed message queue.
<b>SEE ALSO</b>	<code>msgQDistGrpLib</code> , <code>msgQLib</code> , <code>msgQDistLib</code> , <code>distNameLib</code>

---

## msgQDistGrpDelete()

<b>NAME</b>	<b>msgQDistGrpDelete()</b> – delete a distributed message queue from a group (VxFusion Opt.)
<b>SYNOPSIS</b>	<pre><b>STATUS</b> msgQDistGrpDelete (     char *   distGrpName,      /* group containing the queue to be deleted */     MSG_Q_ID msgQId           /* ID of the message queue to delete */ )</pre>
<b>DESCRIPTION</b>	This routine deletes a distributed message queue from a group. <hr/> <b>NOTE:</b> For this release, it is not possible to remove a distributed message queue from a group. <hr/>
<b>AVAILABILITY</b>	This routine is distributed as a component of the unbundled distributed message queues option, VxFusion.
<b>RETURNS</b>	ERROR, always.
<b>ERRNO</b>	S_distLib_NO_OBJECT_DESTROY
<b>SEE ALSO</b>	msgQDistGrpLib

---

## msgQDistGrpShow()

<b>NAME</b>	<b>msgQDistGrpShow()</b> – display all or one group with its members (VxFusion Opt.)
<b>SYNOPSIS</b>	<pre><b>STATUS</b> msgQDistGrpShow (     char * distGrpName      /* name of the group to display or NULL for all */ )</pre>
<b>DESCRIPTION</b>	This routine displays either all distributed message queue groups or a specified group in the group database. For each group displayed on the node, it lists only members added (using <b>msgQDistGrpAdd()</b> ) from the node executing the <b>msgQDistGrpShow()</b> call.  If <i>distGrpName</i> is <b>NULL</b> , all groups and their locally added members are displayed. Otherwise, only the group specified by <i>distGrpName</i> and its locally added members are displayed.

---

**NOTE:** The concept of “locally added” is an important one. All nodes in the system can add groups to a message queue group. However, only those message queues (including remote distributed message queues) that were added to the group from the local node are displayed by this routine.

---

**EXAMPLE**

-> **msgQDistGrpShow(0)**

NAME OF GROUP	GROUP ID	STATE	MEMBER ID	TYPE OF MEMBER
grp1	0x3ff9e3	global	0x3ff98b	distributed msg queue
			0x3ff9fb	distributed msg queue
grp2	0x3ff933	global	0x3ff89b	distributed msg queue
			0x3ff8db	distributed msg queue
			0x3ff94b	distributed msg queue

**AVAILABILITY**

This routine is distributed as a component of the unbundled distributed message queues option, VxFusion.

**RETURNS**

OK, unless name not found.

**ERRNO**

**S\_msgQDistGrpLib\_NO\_MATCH**  
 The group name was not found in the database.

**SEE ALSO**

**msgQDistGrpShow**




---

## msgQDistNumMsgs()

**NAME**

**msgQDistNumMsgs()** – get the number of messages in a distributed message queue (VxFusion Opt.)

**SYNOPSIS**

```
int msgQDistNumMsgs
(
    MSG_Q_ID msgQId,          /* message queue to examine */
    int      overallTimeout /* ticks to wait overall */
)
```

**DESCRIPTION**

This routine returns the number of messages currently queued to a specified distributed message queue.

---

**NOTE:** When **msgQDistNumMsgs()** is called through **msgQNumMsgs()**, *overallTimeout* is set to **WAIT\_FOREVER**. You cannot set *overallTimeout* to **NO\_WAIT** (0) because the process of sending a message from the local node to the remote node always takes a finite amount of time.

---

<b>AVAILABILITY</b>	This routine is distributed as a component of the unbundled distributed message queues option, VxFusion.
<b>RETURNS</b>	The number of messages queued, or <b>ERROR</b> if the operation fails.
<b>ERRNO</b>	<b>S_distLib_OBJ_ID_ERROR</b> The argument <i>msgQId</i> is invalid. <b>S_distLib_UNREACHABLE</b> Could not establish communications with the remote node. <b>S_msgQDistLib_INVALID_TIMEOUT</b> The argument <i>overallTimeout</i> is <b>NO_WAIT</b> .
<b>SEE ALSO</b>	<b>msgQDistLib</b> , <b>msgQLib</b>

---

## msgQDistReceive()

**NAME** msgQDistReceive() – receive a message from a distributed message queue (VxFusion Opt.)

**SYNOPSIS**

```
int msgQDistReceive
(
    MSG_Q_ID msgQId,          /* message queue from which to receive */
    char *   buffer,         /* buffer to receive message */
    UINT     maxNBytes,      /* length of buffer */
    int      msgQTimeout,    /* ticks to wait at the message queue */
    int      overallTimeout /* ticks to wait overall */
)
```

**DESCRIPTION** This routine receives a message from the distributed message queue specified by *msgQId*. The received message is copied into the specified buffer, *buffer*, which is *maxNBytes* in length. If the message is longer than *maxNBytes*, the remainder of the message is discarded (no error indication is returned).

The argument *msgQTimeout* specifies the time in ticks to wait for the queuing of the message. The argument *overallTimeout* specifies the time in ticks to wait for both the sending and queuing of the message. While it is an error to set *overallTimeout* to **NO\_WAIT** (0), **WAIT\_FOREVER** (-1) is allowed for both *msgQTimeout* and *overallTimeout*.

Calling **msgQDistReceive()** on a distributed message group returns an error.

---

**NOTE:** When **msgQDistReceive()** is called through **msgQReceive()**, *msgQTimeout* is set to *timeout* and *overallTimeout* to **WAIT\_FOREVER**.

---

<b>AVAILABILITY</b>	This routine is distributed as a component of the unbundled distributed message queues option, VxFusion.
<b>RETURNS</b>	The number of bytes copied to <i>buffer</i> , or <b>ERROR</b> .
<b>ERRNO</b>	<p><b>S_distLib_OBJ_ID_ERROR</b> The argument <i>msgQId</i> is invalid.</p> <p><b>S_distLib_UNREACHABLE</b> Could not establish communications with the remote node.</p> <p><b>S_msgQLib_INVALID_MSG_LENGTH</b> The argument <i>maxNBytes</i> is less than 0.</p> <p><b>S_msgQDistLib_INVALID_TIMEOUT</b> The argument <i>overallTimeout</i> is <b>NO_WAIT</b>.</p> <p><b>S_msgQDistLib_RMT_MEMORY_SHORTAGE</b> There is not enough memory on the remote node.</p> <p><b>S_objLib_OBJ_UNAVAILABLE</b> The argument <i>msgQTimeout</i> is set to <b>NO_WAIT</b>, and no messages are available.</p> <p><b>S_objLib_OBJ_TIMEOUT</b> No messages were received in <i>msgQTimeout</i> ticks.</p> <p><b>S_msgQDistLib_OVERALL_TIMEOUT</b> There was no response from the remote side in <i>overallTimeout</i> ticks.</p>
<b>SEE ALSO</b>	<b>msgQDistLib</b> , <b>msgQLib</b>

---

## msgQDistSend()

**NAME** `msgQDistSend()` – send a message to a distributed message queue (VxFusion Opt.)

**SYNOPSIS**

```

STATUS msgQDistSend
(
    MSG_Q_ID msgQId,           /* message queue on which to send */
    char *  buffer,         /* message to send */
    UINT    nBytes,         /* length of message */
    int     msgQTimeout,     /* ticks to wait at message queue */
    int     overallTimeout, /* ticks to wait overall */
    int     priority        /* priority */
)

```

<b>DESCRIPTION</b>	<p>This routine sends the message specified by <i>buffer</i> of length <i>nBytes</i> to the distributed message queue or group specified by <i>msgQId</i>.</p> <p>The argument <i>msgQTimeout</i> specifies the time in ticks to wait for the queuing of the message. The argument <i>overallTimeout</i> specifies the time in ticks to wait for both the sending and queuing of the message. While it is an error to set <i>overallTimeout</i> to <b>NO_WAIT</b> (0), <b>WAIT_FOREVER</b> (-1) is allowed for both <i>msgQTimeout</i> and <i>overallTimeout</i>.</p> <p>The <i>priority</i> parameter specifies the priority of the message being sent. It ranges between <b>DIST_MSG_PRI_0</b> (highest priority) and <b>DIST_MSG_PRI_7</b> (lowest priority). A priority of <b>MSG_PRI_URGENT</b> is mapped to <b>DIST_MSG_PRI_0</b>; <b>MSG_PRI_NORMAL</b> is mapped to <b>DIST_MSG_PRI_4</b>. Messages sent with high priorities (<b>DIST_MSG_PRI_0</b> to <b>DIST_MSG_PRI_3</b>) are put to the head of the list of queued messages. Lower priority messages (<b>DIST_MSG_PRI_4</b> to <b>DIST_MSG_PRI_7</b>) are placed at the queue's tail.</p> <hr/> <p><b>NOTE:</b> When <b>msgQDistSend()</b> is called through <b>msgQSend()</b>, <i>msgQTimeout</i> is set to <i>timeout</i> and <i>overallTimeout</i> to <b>WAIT_FOREVER</b>.</p> <hr/>
<b>AVAILABILITY</b>	This routine is distributed as a component of the unbundled distributed message queues option, VxFusion.
<b>RETURNS</b>	OK, or ERROR if the operation fails.
<b>ERRNO</b>	<p><b>S_distLib_OBJ_ID_ERROR</b> The argument <i>msgQId</i> is invalid.</p> <p><b>S_distLib_UNREACHABLE</b> Could not establish communications with the remote node.</p> <p><b>S_msgQDistLib_INVALID_PRIORITY</b> The argument <i>priority</i> is invalid.</p> <p><b>S_msgQDistLib_INVALID_TIMEOUT</b> The argument <i>overallTimeout</i> is <b>NO_WAIT</b>.</p> <p><b>S_msgQDistLib_RMT_MEMORY_SHORTAGE</b> There is not enough memory on the remote node.</p> <p><b>S_objLib_OBJ_UNAVAILABLE</b> The argument <i>msgQTimeout</i> is set to <b>NO_WAIT</b>, and the queue is full.</p> <p><b>S_objLib_OBJ_TIMEOUT</b> The queue is full for <i>msgQTimeout</i> ticks.</p> <p><b>S_msgQLib_INVALID_MSG_LENGTH</b> The argument <i>nBytes</i> is larger than the <i>maxMsgLength</i> set for the message queue.</p> <p><b>S_msgQDistLib_OVERALL_TIMEOUT</b> There was no response from the remote side in <i>overallTimeout</i> ticks.</p>
<b>SEE ALSO</b>	<b>msgQDistLib</b> , <b>msgQLib</b>

---

## msgQDistShowInit( )

<b>NAME</b>	<code>msgQDistShowInit( )</code> – initialize the distributed message queue show package (VxFusion Opt.)
<b>SYNOPSIS</b>	<code>void msgQDistShowInit (void)</code>
<b>DESCRIPTION</b>	This routine initializes the distributed message queue show package. <hr/> <b>NOTE:</b> This routine is called automatically when a target boots using a VxWorks image with VxFusion installed and show routines enabled. <hr/>
<b>AVAILABILITY</b>	This routine is distributed as a component of the unbundled distributed message queues option, VxFusion.
<b>RETURNS</b>	N/A
<b>SEE ALSO</b>	<code>msgQDistShow</code>

---

## msgQEvStart( )

<b>NAME</b>	<code>msgQEvStart( )</code> – start event notification process for a message queue
<b>SYNOPSIS</b>	<pre> STATUS msgQEvStart (     MSG_Q_ID msgQId,           /* msg Q for which to register events */     UINT32  events,           /* 32 possible events */     UINT8   options           /* event-related msg Q options */ ) </pre>
<b>DESCRIPTION</b>	<p>This routine turns on the event notification process for a given message queue. When a message becomes available but not wanted in that particular message queue, the events specified will be sent to the task registered by this function. A task can overwrite its own registration without first invoking <code>msgQEvStop( )</code> or specifying the <code>ALLOW_OVERWRITE</code> option.</p> <p>The <i>options</i> parameter is used for 3 user options:</p> <p><code>EVENTS_SEND_ONCE</code> (0x1) tells the message queue to send the events one time only. Specify if the events are to be sent only once or every time a message arrives until <code>msgQEvStop( )</code> is called.</p>

**EVENTS\_ALLOW\_OVERWRITE** (0x2)

allows subsequent registrations to overwrite the current one. Specify if another task can register itself while the current task is still registered. If so, the current task registration is overwritten without any warning.

**EVENTS\_SEND\_IF\_FREE** (0x4)

tells the registration process to send events if a message is present on the message queue. Specify if events are to be sent right away in the case a message is waiting to be picked up.

If none of those three options is to be used, then the option

**EVENTS\_OPTIONS\_NONE** (0x0)

has to be passed to the *options* parameter.

**RETURNS** OK on success, or **ERROR**.

**ERRNO** **S\_objLib\_OBJ\_ID\_ERROR**  
The message queue ID is invalid.

**S\_eventLib\_ALREADY\_REGISTERED**  
A task is already registered on the message queue.

**S\_intLib\_NOT\_ISR\_CALLABLE**  
Routine has been called from interrupt level.

**S\_eventLib\_EVENTSEND\_FAILED**  
User chose to send events right away and that operation failed.

**S\_eventLib\_ZERO\_EVENTS**  
User passed in a value of zero to the *events* parameter.

**SEE ALSO** **msgQEvLib**, **eventLib**, **msgQLib**, **msgQEvStop()**

---

## msgQEvStop()

**NAME** **msgQEvStop()** – stop event notification process for a message queue

**SYNOPSIS**

```
STATUS msgQEvStop  
(  
    MSG_Q_ID msgQId  
)
```

<b>DESCRIPTION</b>	This routine turns off the event notification process for a given message queue. It thus allows another task to register itself for event notification on that particular message queue.
<b>RETURNS</b>	OK on success, or ERROR.
<b>ERRNO</b>	<p><b>S_objLib_OBJ_ID_ERROR</b> The message queue ID is invalid.</p> <p><b>S_intLib_NOT_ISR_CALLABLE</b> Routine has been called from interrupt level.</p> <p><b>S_eventLib_TASK_NOT_REGISTERED</b> Routine has not been called by registered task.</p>
<b>SEE ALSO</b>	msgQEvLib, eventLib, msgQLib, msgQEvStart()

---

## msgQInfoGet()

**NAME** msgQInfoGet() – get information about a message queue

**SYNOPSIS**

```
STATUS msgQInfoGet
(
    MSG_Q_ID    msgQId,      /* message queue to query */
    MSG_Q_INFO * pInfo      /* where to return msg info */
)
```

**DESCRIPTION** This routine gets information about the state and contents of a message queue. The parameter *pInfo* is a pointer to a structure of type `MSG_Q_INFO` defined in `msgQLib.h` as follows:

```
typedef struct          /* MSG_Q_INFO */
{
    int    numMsgs;      /* OUT: number of messages queued */
    int    numTasks;     /* OUT: number of tasks waiting on msg q */
    int    sendTimeouts; /* OUT: count of send timeouts */
    int    recvTimeouts; /* OUT: count of receive timeouts */
    int    options;      /* OUT: options with which msg q was created */
    int    maxMsgs;      /* OUT: max messages that can be queued */
    int    maxMsgLength; /* OUT: max byte length of each message */
    int    taskIdListMax; /* IN: max tasks to fill in taskIdList */
    int *  taskIdList;   /* PTR: array of task IDs waiting on msg q */
    int    msgListMax;   /* IN: max msgs to fill in msg lists */
    char ** msgPtrList;  /* PTR: array of msg ptrs queued to msg q */
}
```

```
int * msgLenList;      /* PTR: array of lengths of msgs      */  
} MSG_Q_INFO;
```

If a message queue is empty, there may be tasks blocked on receiving. If a message queue is full, there may be tasks blocked on sending. This can be determined as follows:

- If *numMsgs* is 0, then *numTasks* indicates the number of tasks blocked on receiving.

- If *numMsgs* is equal to *maxMsgs*, then *numTasks* is the number of tasks blocked on sending.

- If *numMsgs* is greater than 0 but less than *maxMsgs*, then *numTasks* will be 0.

A list of pointers to the messages queued and their lengths can be obtained by setting *msgPtrList* and *msgLenList* to the addresses of arrays to receive the respective lists, and setting *msgListMax* to the maximum number of elements in those arrays. If either list pointer is *NULL*, no data will be returned for that array.

No more than *msgListMax* message pointers and lengths are returned, although *numMsgs* will always be returned with the actual number of messages queued.

For example, if the caller supplies a *msgPtrList* and *msgLenList* with room for 10 messages and sets *msgListMax* to 10, but there are 20 messages queued, then the pointers and lengths of the first 10 messages in the queue are returned in *msgPtrList* and *msgLenList*, but *numMsgs* will be returned with the value 20.

A list of the task IDs of tasks blocked on the message queue can be obtained by setting *taskIdList* to the address of an array to receive the list, and setting *taskIdListMax* to the maximum number of elements in that array. If *taskIdList* is *NULL*, then no task IDs are returned. No more than *taskIdListMax* task IDs are returned, although *numTasks* will always be returned with the actual number of tasks blocked.

For example, if the caller supplies a *taskIdList* with room for 10 task IDs and sets *taskIdListMax* to 10, but there are 20 tasks blocked on the message queue, then the IDs of the first 10 tasks in the blocked queue will be returned in *taskIdList*, but *numTasks* will be returned with the value 20.

Note that the tasks returned in *taskIdList* may be blocked for either send or receive. As noted above this can be determined by examining *numMsgs*.

The variables *sendTimeouts* and *recvTimeouts* are the counts of the number of times **msgQSend()** and **msgQReceive()** respectively returned with a timeout.

The variables *options*, *maxMsgs*, and *maxMsgLength* are the parameters with which the message queue was created.

---

**WARNING:** The information returned by this routine is not static and may be obsolete by the time it is examined. In particular, the lists of task IDs and/or message pointers may no

longer be valid. However, the information is obtained atomically, thus it will be an accurate snapshot of the state of the message queue at the time of the call. This information is generally used for debugging purposes only.

---

**WARNING:** The current implementation of this routine locks out interrupts while obtaining the information. This can compromise the overall interrupt latency of the system. Generally this routine is used for debugging purposes only.

---

**RETURNS** OK or ERROR.

**ERRNO** S\_distLib\_NOT\_INITIALIZED, S\_smObjLib\_NOT\_INITIALIZED, S\_objLib\_OBJ\_ID\_ERROR

**SEE ALSO** msgQShow

---

## msgQNumMsgs()

**NAME** msgQNumMsgs() – get the number of messages queued to a message queue

**SYNOPSIS**

```
int msgQNumMsgs
(
    MSG_Q_ID msgQId          /* message queue to examine */
)
```

**DESCRIPTION** This routine returns the number of messages currently queued to a specified message queue.

**RETURNS** The number of messages queued, or ERROR.

**ERRNO** S\_distLib\_NOT\_INITIALIZED, S\_smObjLib\_NOT\_INITIALIZED, S\_objLib\_OBJ\_ID\_ERROR

**SEE ALSO** msgQLib, msgQSmLib



---

## msgQReceive()

**NAME** msgQReceive() – receive a message from a message queue

**SYNOPSIS**

```
int msgQReceive
(
    MSG_Q_ID msgQId,          /* message queue from which to receive */
    char *   buffer,         /* buffer to receive message */
    UINT     maxNBytes,     /* length of buffer */
    int      timeout        /* ticks to wait */
)
```

**DESCRIPTION** This routine receives a message from the message queue *msgQId*. The received message is copied into the specified *buffer*, which is *maxNBytes* in length. If the message is longer than *maxNBytes*, the remainder of the message is discarded (no error indication is returned).

The *timeout* parameter specifies the number of ticks to wait for a message to be sent to the queue, if no message is available when **msgQReceive()** is called. The *timeout* parameter can also have the following special values:

**NO\_WAIT** (0)

return immediately, whether a message has been received or not.

**WAIT\_FOREVER** (-1)

never time out.

---

**WARNING:** This routine must not be called by interrupt service routines.

---

**RETURNS** The number of bytes copied to *buffer*, or **ERROR**.

**ERRNO** S\_distLib\_NOT\_INITIALIZED, S\_smObjLib\_NOT\_INITIALIZED, S\_objLib\_OBJ\_ID\_ERROR, S\_objLib\_OBJ\_DELETED, S\_objLib\_OBJ\_UNAVAILABLE, S\_objLib\_OBJ\_TIMEOUT, S\_msgQLib\_INVALID\_MSG\_LENGTH, S\_intLib\_NOT\_ISR\_CALLABLE

**SEE ALSO** msgQLib, msgQSmLib

---

## msgQSend()

**NAME** msgQSend() – send a message to a message queue

**SYNOPSIS**

```
STATUS msgQSend
(
    MSG_Q_ID msgQId,          /* message queue on which to send */
    char *   buffer,         /* message to send */
    UINT     nBytes,         /* length of message */
    int      timeout,        /* ticks to wait */
    int      priority        /* MSG_PRI_NORMAL or MSG_PRI_URGENT */
)
```

**DESCRIPTION** This routine sends the message in *buffer* of length *nBytes* to the message queue *msgQId*. If any tasks are already waiting to receive messages on the queue, the message is immediately delivered to the first waiting task. If no task is waiting to receive messages, the message is saved in the message queue and if a task has previously registered to receive events from the message queue, these events are sent in the context of this call. This may result in the unpending of the task waiting for the events. If the message queue fails to send events and if it was created using the `MSG_Q_EVENTSEND_ERR_NOTIFY` option, `ERROR` is returned even though the send operation was successful.

The *timeout* parameter specifies the number of ticks to wait for free space if the message queue is full. The *timeout* parameter can also have the following special values:

`NO_WAIT (0)`  
return immediately, even if the message has not been sent.

`WAIT_FOREVER (-1)`  
never time out.

The *priority* parameter specifies the priority of the message being sent. The possible values are:

`MSG_PRI_NORMAL (0)`  
normal priority; add the message to the tail of the list of queued messages.

`MSG_PRI_URGENT (1)`  
urgent priority; add the message to the head of the list of queued messages.

### USE BY INTERRUPT SERVICE ROUTINES

This routine can be called by interrupt service routines as well as by tasks. This is one of the primary means of communication between an interrupt service routine and a task. When called from an interrupt service routine, *timeout* must be `NO_WAIT`.

**RETURNS** OK on success or `ERROR` otherwise.

<b>ERRNO</b>	<b>S_distLib_NOT_INITIALIZED</b> Distributed objects message queue library (VxFusion) not initialized.
	<b>S_smObjLib_NOT_INITIALIZED</b> Shared memory message queue library (VxMP Opt.) not initialized.
	<b>S_objLib_OBJ_ID_ERROR</b> Invalid message queue ID.
	<b>S_objLib_OBJ_DELETED</b> Message queue deleted while calling task was pended.
	<b>S_objLib_OBJ_UNAVAILABLE</b> No free buffer space when <b>NO_WAIT</b> timeout specified.
	<b>S_objLib_OBJ_TIMEOUT</b> Timeout occurred while waiting for buffer space.
	<b>S_msgQLib_INVALID_MSG_LENGTH</b> Message length exceeds limit.
	<b>S_msgQLib_NON_ZERO_TIMEOUT_AT_INT_LEVEL</b> Called from ISR with non-zero timeout.
	<b>S_eventLib_EVENTSEND_FAILED</b> Message queue failed to send events to registered task. This <b>errno</b> value can only exist if the message queue was created with the <b>MSG_Q_EVENTSEND_ERR_NOTIFY</b> option.
<b>SEE ALSO</b>	<b>msgQLib, msgQSmLib, msgQEvStart()</b>

---

## msgQShow()

<b>NAME</b>	<b>msgQShow()</b> – show information about a message queue
<b>SYNOPSIS</b>	<pre><b>STATUS</b> msgQShow (     <b>MSG_Q_ID</b> msgQId,          /* message queue to display */     <b>int</b>      level           /* 0 = summary, 1 = details */ )</pre>
<b>DESCRIPTION</b>	<p>This routine displays the state and optionally the contents of a message queue. A summary of the state of the message queue is displayed as follows:</p> <pre><b>Message Queue Id</b>      : 0x3f8c20 <b>Task Queuing</b>         : FIFO <b>Message Byte Len</b>     : 150 <b>Messages Max</b>        : 50</pre>

```

Messages Queued      : 0
Receivers Blocked   : 1
Send timeouts       : 0
Receive timeouts    : 0
Options              : 0x1          MSG_Q_FIFO
VxWorks Events
-----
Registered Task     : 0x3f5c70 (t1)
Event(s) to Send    : 0x1
Options              : 0x7          EVENTS_SEND_ONCE
                                   EVENTS_ALLOW_OVERWRITE
                                   EVENTS_SEND_IF_FREE

```

If *level* is 1, then more detailed information will be displayed. If messages are queued, they will be displayed as follows:

```

Messages queued:
#   address length value
1 0x123eb204   4 0x00000001 0x12345678

```

If tasks are blocked on the queue, they will be displayed as follows:

```

Receivers blocked:
NAME      TID     PRI  DELAY
-----
tExcTask  3fd678  0    21

```

**RETURNS** OK or ERROR.

**ERRNO** S\_distLib\_NOT\_INITIALIZED, S\_smObjLib\_NOT\_INITIALIZED

**SEE ALSO** msgQShow, *VxWorks Programmer's Guide: Target Shell*, *windsh*, *Tornado User's Guide: Shell*

---

## msgQShowInit()

**NAME** msgQShowInit() – initialize the message queue show facility

**SYNOPSIS** void msgQShowInit (void)

**DESCRIPTION** This routine links the message queue show facility into the VxWorks system. It is called automatically when the message queue show facility is configured into VxWorks using either of the following methods:

- If you use the configuration header files, define `INCLUDE_SHOW_ROUTINES` in `config.h`.
- If you use the Tornado project facility, select `INCLUDE_MSG_Q_SHOW`.

**RETURNS** N/A

**SEE ALSO** msgQShow

---

## msgQSmCreate()

**NAME** msgQSmCreate() – create and initialize a shared memory message queue (VxMP Opt.)

**SYNOPSIS**

```
MSG_Q_ID msgQSmCreate
(
    int maxMsgs,           /* max messages that can be queued */
    int maxMsgLength,     /* max bytes in a message */
    int options           /* message queue options */
)
```

**DESCRIPTION** This routine creates a shared memory message queue capable of holding up to *maxMsgs* messages, each up to *maxMsgLength* bytes long. It returns a message queue ID used to identify the created message queue. The queue can only be created with the option `MSG_Q_FIFO (0)`, thus queuing pending tasks in FIFO order.

The global message queue identifier returned can be used directly by generic message queue handling routines in `msgQLib` -- `msgQSend()`, `msgQReceive()`, and `msgQNumMsgs()` -- and by the show routines `show()` and `msgQShow()`.

If there is insufficient memory to store the message queue structure in the shared memory message queue partition or if the shared memory system pool cannot handle the requested message queue size, shared memory message queue creation will fail with `errno` set to `S_memLib_NOT_ENOUGH_MEMORY`. This problem can be solved by incrementing the value of `SM_OBJ_MAX_MSG_Q` and/or the shared memory objects dedicated memory size `SM_OBJ_MEM_SIZE`.

Before this routine can be called, the shared memory objects facility must be initialized (see `msgQSmLib`).

**AVAILABILITY** This routine is distributed as a component of the unbundled shared memory objects support option, VxMP.

**RETURNS** `MSG_Q_ID`, or NULL if error.

**ERRNO** `S_memLib_NOT_ENOUGH_MEMORY`, `S_intLib_NOT_ISR_CALLABLE`,  
`S_msgQLib_INVALID_QUEUE_TYPE`, `S_smObjLib_LOCK_TIMEOUT`

**SEE ALSO** `msgQSmLib`, `smObjLib`, `msgQLib`, `msgQShow`

---

## **munlock()**

**NAME** `munlock()` – unlock specified pages (POSIX)

**SYNOPSIS**

```
int munlock
(
    const void * addr,
    size_t      len
)
```

**DESCRIPTION** This routine unlocks specified pages from being memory resident.

**RETURNS** 0 (OK) always.

**ERRNO** N/A

**SEE ALSO** `mmanPxLib`

---

## **munlockall()**

**NAME** `munlockall()` – unlock all pages used by a process (POSIX)

**SYNOPSIS**

```
int munlockall (void)
```

**DESCRIPTION** This routine unlocks all pages used by a process from being memory resident.

**RETURNS** 0 (OK) always.

**ERRNO** N/A

**SEE ALSO** `mmanPxLib`

---

## **muxAddressForm()**

**NAME** `muxAddressForm()` – form a frame with a link-layer address

**SYNOPSIS**

```
M_BLK_ID muxAddressForm  
(  
    void *   pCookie,           /* protocol/device binding from muxBind() */  
    M_BLK_ID pMblk,           /* structure to contain packet */  
    M_BLK_ID pSrcAddr,       /* structure containing source address */  
    M_BLK_ID pDstAddr        /* structure containing destination address */  
)
```

**DESCRIPTION** Use this routine to create a frame with an appropriate link-layer address. As input, this function expects the source address, the destination address, and the data you want to include in the frame. When control returns from the `muxAddressForm()` call, the `pMblk` parameter references a frame ready for transmission. Internally, `muxAddressForm()` either prepended the link-layer header to the data buffer supplied in `pMblk` (if there was enough room) or it allocated a new `mBlk-clBlk`-cluster and prepended the new `mBlk` to the `mBlk` chain supplied in `pMblk`.

---

**NOTE:** You should set the `pDstAddr.mBlkHdr.reserved` field to the network service type.

---

*pCookie*

Expects the cookie returned from the `muxBind()`. This cookie indicates the device to which the MUX has bound this protocol.

*pMblk*

Expects a pointer to the `mBlk` structure that contains the packet.

*pSrcAddr*

Expects a pointer to the `mBlk` that contains the source address.

*pDstAddr*

Expects a pointer to the `mBlk` that contains the destination address.

---

**NOTE:** This routine is used only with ENDS, and is not needed for NPT drivers.

---

### **VXWORKS AE PROTECTION DOMAINS**

Under VxWorks AE, you can call `muxAddressForm()` from within the kernel protection domain only, and the data referenced in the `pCookie` parameter must reside in the kernel protection domain. In addition, the returned `M_BLK_ID` is valid in the kernel protection domain only. This restriction does not apply under non-AE versions of VxWorks.

**RETURNS** `M_BLK_ID` or `NULL`.

**ERRNO**            **S\_muxLib\_NO\_DEVICE**

**SEE ALSO**        **muxLib**

---

## **muxAddrResFuncAdd()**

**NAME**            **muxAddrResFuncAdd()** – replace the default address resolution function

**SYNOPSIS**        **STATUS** **muxAddrResFuncAdd**

```
(  
    long    ifType,      /* Media interface type, typically from m2Lib.h */  
    long    protocol,   /* Service type, for instance from RFC 1700 */  
    FUNCPTR addrResFunc /* Function to call. */  
)
```

**DESCRIPTION**    Use this routine to register an address resolution function for an interface-type/protocol pair. You must call **muxAddrResFuncAdd()** prior to calling the protocol's **protocolAttach()** routine. If the driver registers itself as an Ethernet driver, you do not need to call this routine. VxWorks automatically assigns **arpresolve()** to registered Ethernet devices. The **muxAddrResFuncAdd()** functionality is intended for using the VxWorks network stack with non-Ethernet drivers that require address resolution.

### *ifType*

Expects a media interface or network driver type, such as can be found in **m2Lib.h**. If using the END model, the *ifType* argument is restricted to the values in **m2Lib.h**. In the NPT model, this restriction does not apply.

### *protocol*

Expects a network service or protocol type, such as can be found in RFC 1700. Look for the values under ETHER TYPES. For example, Internet IP would be identified as 2048 (0x800 hexadecimal). If using the END model, *protocol* is restricted to the values in RFC 1700. In the NPT model, this restriction does not apply.

### *addrResFunc*

Expects a pointer to an address resolution function for this combination of driver type and service type. The prototype of your replacement address resolution function must match that of **arpresolve()**:

```
int arpresolve  
(  
    struct arpcom *   ac,  
    struct rtentry *  rt,  
    struct mbuf *     m,  
    struct sockaddr * dst,
```

```
    u_char *    desten  
    )
```

This function returns one upon success, which indicates that *desten* has been updated with the necessary data-link layer information and that the IP sublayer output function can transmit the packet.

This function returns zero if it cannot resolve the address immediately. In the default **arpresolve()** implementation, resolving the address immediately means **arpresolve()** was able to find the address in its table of results from previous ARP requests. Returning zero indicates that the table did not contain the information but that the packet has been stored and that an ARP request has been queued.

If the ARP request times out, the packet is dropped. If the ARP request completes successfully, processing that event updates the local ARP table and resubmits the packet to the IP sublayer's output function for transmission. This time, the **arpresolve()** call will return one.

What is essential to note here is that **arpresolve()** did not wait for the ARP request to complete before returning. If you replace the default **arpresolve()** function, you must make sure your function returns as soon as possible and that it never blocks. Otherwise, you block the IP sublayer from transmitting other packets out through the interface for which this packet was queued. You must also make sure that your **arpresolve()** function takes responsibility for the packet if it returns zero. Otherwise, the packet is dropped.

#### VXWORKS AE PROTECTION DOMAINS

Under VxWorks AE, you can call **muxAddrResFuncAdd()** from within the kernel protection domain only, and the data referenced in the *addrResFunc* parameter must reside in the kernel protection domain. This restriction does not apply under non-AE versions of VxWorks.

**RETURNS** OK, or ERROR.

**SEE ALSO** [muxLib](#)

---

## **muxAddrResFuncDel()**

**NAME** **muxAddrResFuncDel()** – delete an address resolution function

**SYNOPSIS**

```
STATUS muxAddrResFuncDel  
(  
    long ifType,          /* ifType of function you want to delete */  
    long protocol        /* protocol from which to delete the function */  
)
```

**DESCRIPTION** This function deletes the address resolution function registered for the specified interface-protocol pair. If using the NPT architecture, the *ifType* and *protocol* arguments are not restricted to the **m2Lib.h** or RFC 1700 values.

*ifType*

Expects a media interface or network driver type. For an END driver, use the values specified in **m2Lib.h**.

*protocol*

Expects a network service or protocol type. For example, Internet IP would be identified as 2048 (0x800 hexadecimal). This value can be found in RFC 1700 under the heading, ETHER TYPES.

**VXWORKS AE PROTECTION DOMAINS**

Under VxWorks AE, you can call **muxAddrResFuncDel()** from within the kernel protection domain only. This restriction does not apply under non-AE versions of VxWorks.

**RETURNS** OK or ERROR.

**SEE ALSO** **muxLib**




---

## **muxAddrResFuncGet()**

**NAME** **muxAddrResFuncGet()** – get the address resolution function for *ifType*/protocol

**SYNOPSIS** **FUNCPTR muxAddrResFuncGet**

```
(
    long ifType,           /* ifType from m2Lib.h */
    long protocol         /* protocol from RFC 1700 */
)
```

**DESCRIPTION** This routine gets a pointer to the registered address resolution function for the specified interface-protocol pair. If no such function exists, **muxAddrResFuncGet()** returns NULL.

*ifType*

Expects a media interface or network driver type, such as those found in **m2Lib.h**. If using the END model, the *ifType* argument is restricted to the **m2Lib.h** values. In the NPT model, this restriction does not apply.

*protocol*

Expects a network service or protocol type such as those found in RFC 1700. Look for the values under ETHER TYPES. For example, Internet IP would be identified as 2048 (0x800 hexadecimal). If using the END model, the *protocol* argument is restricted to the RFC 1700 values. In the NPT model, this restriction does not apply.

#### VXWORKS AE PROTECTION DOMAINS

Under VxWorks AE, you can call **muxAddrResFuncGet()** from within the kernel protection domain only. In addition, the returned FUNCPTR is valid in the kernel protection domain only. This restriction does not apply under non-AE versions of VxWorks.

**RETURNS** FUNCPTR to the routine or NULL.

**SEE ALSO** muxLib

---

## muxBind()

**NAME** muxBind() – create a binding between a network service and an END

#### SYNOPSIS

```
void * muxBind
(
  char * pName,           /* interface name, for example, ln, ei,... */
  int   unit,            /* unit number */
  BOOL (* stackRcvRtn) (void* , long, M_BLK_ID, LL_HDR_INFO * , void* ),
                        /* receive function to be called. */
  STATUS (* stackShutdownRtn) (void* , void* ),
                        /* routine to call to shutdown the stack */
  STATUS (* stackTxRestartRtn) (void* , void* ),
                        /* routine to tell the stack it can transmit */
  void (* stackErrorRtn) (END_OBJ* , END_ERR* , void* ),
                        /* routine to call on an error. */
  long  type,            /* protocol type from RFC1700 and many */
                        /* other sources (for example, 0x800 is IP) */
  char * pProtoName,     /* string name for protocol */
  void * pSpare          /* per protocol spare pointer */
)
```

**DESCRIPTION** A network service uses this routine to bind to an END specified by the *pName* and *unit* arguments (for example, ln and 0, ln and 1, or ei and 0).

---

**NOTE:** This routine should only be used to bind to drivers that use the old END driver callback function prototypes. NPT drivers, or END drivers that use the newer callback function prototypes, should use **muxTkBind()** instead. See the *Network Protocol Toolkit Programmer's Guide* for more information on when to use **muxBind()** and **muxTkBind()**.

---

The *type* argument assigns a network service to one of several classes. Standard services receive the portion of incoming data associated with *type* values from RFC 1700. Only one service for each RFC 1700 type value may be bound to an END.

Services with *type* **MUX\_PROTO\_SNARF** provide a mechanism for bypassing the standard services for purposes such as firewalls. These services will get incoming packets before any of the standard services.

Promiscuous services with *type* **MUX\_PROTO\_PROMISC** receive any packets not consumed by the snarf or standard services.

The MUX allows multiple snarf and promiscuous services but does not coordinate between them. It simply delivers available packets to each service in FIFO order. Services that consume packets may prevent “downstream” services from receiving data if the desired packets overlap.

An output service (with *type* **MUX\_PROTO\_OUTPUT**) receives outgoing data before it is sent to the device. This service type allows two network services to communicate directly and provides a mechanism for loop-back testing. Only one output service is supported for each driver.

The MUX calls the registered *stackRcvRtn* whenever it receives a packet of the appropriate type. If that routine returns **TRUE**, the packet is not offered to any remaining services (or to the driver in the case of output services). A service (including an output service) may return **FALSE** to examine a packet without consuming it. See the description of a **stackRcvRtn()** in the *Network Protocol Toolkit Programmer's Guide* for additional information about the expected behavior of that routine.

The *stackShutdownRtn* argument provides a function that the MUX can use to shut down the service. See the *Network Protocol Toolkit Programmer's Guide* for a description of how to write such a routine.

The *pProtoName* argument provides the name of the service as a character string. A service name is assigned internally if the argument is **NULL**.

The *pSpare* argument registers a pointer to data defined by the service. The MUX includes this argument in calls to the call back routines from this service.

#### **VXWORKS AE PROTECTION DOMAINS**

Under VxWorks AE, you can call **muxBind()** from within the kernel protection domain only, and the data referenced in the *stackRcvRtn*, *stackShutdownRtn*, *stackTxRestartRtn*, *stackErrorRtn* and *pSpare* parameters must reside in the kernel protection domain. In addition, the returned void pointer is valid in the kernel protection domain only. This restriction does not apply under non-AE versions of VxWorks.

<b>RETURNS</b>	A cookie identifying the binding between service and driver; or <b>NULL</b> , if an error occurred.
<b>ERRNO</b>	<b>S_muxLib_NO_DEVICE</b> , <b>S_muxLib_ALREADY_BOUND</b> , <b>S_muxLib_ALLOC_FAILED</b>
<b>SEE ALSO</b>	<b>muxLib</b>

---

## **muxDevExists()**

**NAME** **muxDevExists()** – tests whether a device is already loaded into the MUX

**SYNOPSIS**

```
BOOL muxDevExists
(
    char * pName,          /* string containing a device name (ln, ei, ...)*/
    int   unit            /* unit number */
)
```

**DESCRIPTION** This routine takes a string device name (for example, ln or ei) and a unit number. If this device is already known to the MUX, it returns TRUE. Otherwise, this routine returns FALSE.

*pName*

Expects a pointer to a string containing the device name

*unit*

Expects the unit number of the device

### **VXWORKS AE PROTECTION DOMAINS**

Under VxWorks AE, you can call **muxDevExists()** from within the kernel protection domain only, and the data referenced in the *pName* parameter must reside in the kernel protection domain. This restriction does not apply under non-AE versions of VxWorks.

**RETURNS** TRUE, if the device exists; else FALSE.

**SEE ALSO** **muxLib**

---

## **muxDevLoad()**

**NAME** **muxDevLoad()** – load a driver into the MUX

**SYNOPSIS**

```
void * muxDevLoad
(
    int   unit,          /* unit number of device */
    END_OBJ * (* endLoad) (char* , void* ),
                                /* load function of the driver */
    char * pInitString,  /* init string for this driver */
    BOOL  loaning,      /* we loan buffers */
    void * pBSP          /* for BSP group */
)
```

**DESCRIPTION** The **muxDevLoad()** routine loads a network driver into the MUX. Internally, this routine calls the specified *endLoad* routine to initialize the software state of the device. After the device is initialized, you must call **muxDevStart()** to start the device.

*unit*

Expects the unit number of the device.

*endLoad*

Expects a pointer to the network driver's **endLoad()** or **nptLoad()** entry point.

*pInitString*

Expects a pointer to an initialization string, typically a colon-delimited list of options. The **muxDevLoad()** routine passes this along blindly to the *endLoad* function.

*loaning*

Currently unused.

*pBSP*

The MUX blindly passes this argument to the driver, which may or may not use it. Some BSPs use this parameter to pass in tables of functions that the driver can use to deal with the particulars of the BSP.

#### **VXWORKS AE PROTECTION DOMAINS**

Under VxWorks AE, you can call **muxDevLoad()** from within the kernel protection domain only, and the data referenced in the *endLoad* and *pBSP* parameters must reside in the kernel protection domain. In addition, the returned void pointer is valid in the kernel protection domain only. This restriction does not apply under non-AE versions of VxWorks.

**RETURNS** A cookie representing the new device, or NULL if an error occurred.

**ERRNO** **S\_muxLib\_LOAD\_FAILED**

**SEE ALSO** **muxLib**

---

## **muxDevStart()**

**NAME** **muxDevStart()** – start a device by calling its start routine

**SYNOPSIS**

```
STATUS muxDevStart
(
    void * pCookie          /* device identifier from muxDevLoad() routine */
)
```

**DESCRIPTION** This routine starts a device that has already been initialized and loaded into the MUX with **muxDevLoad()**. **muxDevStart()** activates the network interfaces for a device, and calls the device's **endStart()** or **nptStart()** routine, which registers the driver's interrupt service routine and does whatever else is needed to allow the device to handle receiving and transmitting. This call to **endStart()** or **nptStart()** puts the device into a running state.

*pCookie*

Expects the pointer returned as the function value of the **muxDevLoad()** call for this device. This pointer identifies the device.

**VXWORKS AE PROTECTION DOMAINS**

Under VxWorks AE, you can call **muxDevStart()** from within the kernel protection domain only, and the data referenced in the *pCookie* parameter must reside in the kernel protection domain. This restriction does not apply under non-AE versions of VxWorks.

**RETURNS** **OK**; **ENETDOWN**, if *pCookie* does not represent a valid device; or **ERROR**, if the start routine for the device fails.

**ERRNO** **S\_muxLib\_NO\_DEVICE**

**SEE ALSO** **muxLib**

---

## **muxDevStop()**

**NAME** **muxDevStop()** – stop a device by calling its stop routine

**SYNOPSIS**

```
STATUS muxDevStop  
(  
    void * pCookie          /* device identifier from muxDevLoad() routine */  
)
```

**DESCRIPTION** This routine stops the device specified in *pCookie*. **muxDevStop()** calls the device's **endStop()** or **nptStop()** routine.

*pCookie*

Expects the cookie returned as the function value of the **muxDevLoad()** call for this device. This cookie identifies the device.

**VXWORKS AE PROTECTION DOMAINS**

Under VxWorks AE, you can call **muxDevStop()** from within the kernel protection domain only, and the data referenced in the *pCookie* parameter must reside in the kernel protection domain. This restriction does not apply under non-AE versions of VxWorks.

**RETURNS** OK; ENETDOWN, if *pCookie* does not represent a valid device; or ERROR, if the **endStop()** or **nptStop()** routine for the device fails.

**ERRNO** S\_muxLib\_NO\_DEVICE

**SEE ALSO** muxLib

---

## **muxDevUnload()**

**NAME** muxDevUnload() – unloads a device from the MUX

**SYNOPSIS**

```
STATUS muxDevUnload
(
    char * pName,           /* a string containing the name of the */
                           /* device for example, ln or ei */
    int    unit            /* the unit number */
)
```

**DESCRIPTION** This routine unloads a device from the MUX. This breaks any network connections that use the device. When this routine is called, each service bound to the device disconnects from it with the **stackShutdownRtn()** routine that was registered by the service. The **stackShutdownRtn()** should call **muxUnbind()** to detach from the device. Then, **muxDevUnload()** calls the device's **endUnload()** or **nptUnload()** routine.

*pName*

Expects a pointer to a string containing the name of the device, for example **ln** or **ei**

*unit*

Expects the unit number of the device indicated by *pName*

**VXWORKS AE PROTECTION DOMAINS**

Under VxWorks AE, you can call **muxDevUnLoad()** from within the kernel protection domain only. This restriction does not apply under non-AE versions of VxWorks.

**RETURNS** OK, on success; ERROR, if the specified device was not found or some other error occurred; or the error value returned by the driver's **unload()** routine.

**ERRNO** S\_muxLib\_UNLOAD\_FAILED, S\_muxLib\_NO\_DEVICE

**SEE ALSO** muxLib



---

## muxIoctl()

**NAME** `muxIoctl()` – send control information to the MUX or to a device

**SYNOPSIS**

```
STATUS muxIoctl
(
    void * pCookie,          /* service/device binding from */
                              /* muxBind()/muxTkBind() */
    int cmd,                /* command to pass to ioctl */
    caddr_t data             /* data need for command in cmd */
)
```

**DESCRIPTION** This routine gives the service access to the network driver's control functions. The MUX itself can implement some of the standard control functions, so not all commands necessarily pass down to the device. Otherwise, both command and data pass to the device without modification.

Typical uses of `muxIoctl()` include commands to start, stop, or reset the network interface, or to add or configure MAC and network addresses.

`pCookie`

Expects the cookie returned from `muxBind()` or `muxTkBind()`. This cookie indicates the device to which this service is bound.

`cmd`

Expects a value indicating the control command you want to execute. For valid `cmd` values, see the description of the `endIoctl()` and `nptIoctl()` routines provided in the *Network Protocol Toolkit Programmer's Guide*.

`data`

Expects the data or a pointer to the data needed to carry out the command specified in `cmd`.

### VXWORKS AE PROTECTION DOMAINS

Under VxWorks AE, you can call `muxIoctl()` from within the kernel protection domain only, and the data referenced in the `pCookie` and `data` parameters must reside in the kernel protection domain. This restriction does not apply under non-AE versions of VxWorks.

**RETURNS** `OK`; `ENETDOWN`, if `pCookie` does not represent a bound device; or `ERROR`, if the command fails.

**ERRNO** `S_muxLib_NO_DEVICE`

**SEE ALSO** `muxLib`

---

## muxLibInit()

**NAME** `muxLibInit()` – initialize global state for the MUX

**SYNOPSIS** `STATUS muxLibInit (void)`

**DESCRIPTION** This routine initializes all global states for the MUX.

**RETURNS** OK or ERROR.

**SEE ALSO** `muxLib`

---

## muxLinkHeaderCreate()

**NAME** `muxLinkHeaderCreate()` – attach a link-level header to a packet

**SYNOPSIS**

```
M_BLK_ID muxLinkHeaderCreate
(
    void *   pCookie,           /* protocol/device binding from muxBind() */
    M_BLK_ID pPacket,          /* structure containing frame contents */
    M_BLK_ID pSrcAddr,         /* structure containing source address */
    M_BLK_ID pDstAddr,         /* structure containing destination address */
    BOOL     bcastFlag         /* use broadcast destination (if available)? */
)
```

**DESCRIPTION** This routine constructs a link-level header using the source address of the device indicated by the *pCookie* argument as returned from the `muxBind()` routine.

The *pDstAddr* argument provides an `M_BLK_ID` buffer containing the link-level destination address. Alternatively, the *bcastFlag* argument, if `TRUE`, indicates that the routine should use the link-level broadcast address, if available for the device. Although other information contained in the *pDstAddr* argument must be accurate, the address data itself is ignored in that case.

The *pPacket* argument contains the contents of the resulting link-level frame. This routine prepends the new link-level header to the initial `mBlk` in that network packet if space is available or allocates a new `mBlk-clBlk`-cluster triplet and prepends it to the `mBlk` chain. When construction of the header is complete, it returns an `M_BLK_ID` that points to the initial `mBlk` in the assembled link-level frame.

**RETURNS** `M_BLK_ID` or `NULL`.



---

## muxMCastAddrDel()

**NAME** `muxMCastAddrDel()` – delete a multicast address from a device’s multicast table

**SYNOPSIS**

```
STATUS muxMCastAddrDel
(
    void * pCookie,          /* binding instance from muxBind() or */
                           /* muxTkBind() */
    char * pAddress         /* Address to delete from the table. */
)
```

**DESCRIPTION** This routine deletes an address from the multicast table maintained by a device by calling that device’s `endMCastAddrDel()` or `nptMCastAddrDel()` routine.

If the device does not support multicasting, `muxMCastAddrAdd()` will return **ERROR** and **errno** will be set to **ENOTSUP** (assuming the driver has been written properly).

*pCookie*

Expects the cookie returned from `muxBind()` or `muxTkBind()` call. This cookie identifies the device to which the MUX bound this service.

*pAddress*

Expects a pointer to a character string containing the address you want to delete.

### VXWORKS AE PROTECTION DOMAINS

Under VxWorks AE, you can call `muxMCastAddrDel()` from within the kernel protection domain only, and the data referenced in the *pCookie* parameter must reside in the kernel protection domain. This restriction does not apply under non-AE versions of VxWorks.

**RETURNS** **OK**; **ENETDOWN**, if *pCookie* does not represent a valid driver; or **ERROR**, if the driver’s registered `endMCastAddrDel()` or `nptMCastAddrDel()` functions fail.

**ERRNO** `S_muxLib_NO_DEVICE`

**SEE ALSO** `muxLib`

---

## **muxMCastAddrGet()**

**NAME** `muxMCastAddrGet()` – get the multicast address table from the MUX/Driver

**SYNOPSIS**

```
int muxMCastAddrGet
(
    void *      pCookie,    /* binding instance from muxBind() or */
                          /* muxTkBind() */
    MULTI_TABLE * pTable    /* pointer to a table to be filled and */
                          /* returned. */
)
```

**DESCRIPTION** This routine writes the list of multicast addresses for a specified device into a buffer. To get this list, it calls the driver's own `endMCastAddrGet()` or `nptMCastAddrGet()` routine.

`pCookie`

Expects the cookie returned from `muxBind()` or `muxTkBind()` call. This cookie indicates the device to which the MUX has bound this service.

`pTable`

Expects a pointer to a `MULTI_TABLE` structure. You must have allocated this structure at some time before the call to `muxMCastAddrGet()`. The `MULTI_TABLE` structure is defined in `end.h` as:

```
typedef struct multi_table
{
    int      tableLen; /* length of table in bytes */
    char *  pTable;   /* pointer to entries */
} MULTI_TABLE;
```

### **VXWORKS AE PROTECTION DOMAINS**

Under VxWorks AE, you can call `muxMCastAddrGet()` from within the kernel protection domain only, and the data referenced in the `pCookie` parameter must reside in the kernel protection domain. This restriction does not apply under non-AE versions of VxWorks.

**RETURNS** `OK`; `ENETDOWN`, if `pCookie` does not represent a valid driver; or `ERROR`, if the driver's registered `endMCastAddrGet()` or `nptMCastAddrGet()` routines fail.

**ERRNO** `S_muxLib_NO_DEVICE`

**SEE ALSO** `muxLib`

---

## muxPacketAddrGet()

**NAME** `muxPacketAddrGet()` – get addressing information from a packet

**SYNOPSIS**

```
STATUS muxPacketAddrGet
(
    void * pCookie,           /* protocol/device binding from muxBind() */
    M_BLK_ID pMblk,          /* structure to contain packet */
    M_BLK_ID pSrcAddr,       /* structure containing source address */
    M_BLK_ID pDstAddr,       /* structure containing destination address */
    M_BLK_ID pESrcAddr,      /* structure containing the end source */
    M_BLK_ID pEDstAddr       /* structure containing the end destination */
)
```

**DESCRIPTION** The routine returns the immediate source, immediate destination, ultimate source, and ultimate destination addresses from the packet pointed to in the first `M_BLK_ID`. This routine makes no attempt to extract that information from the packet directly. Instead, it passes the packet to the driver call that knows how to interpret the packets it has received.

*pCookie*

Expects the cookie returned from the `muxBind()` call. This cookie indicates the device to which the MUX bound this service.

*pMblk*

Expects an `M_BLK_ID` representing packet data from which the addressing information is to be extracted

*pSrcAddr*

Expects NULL or an `M_BLK_ID` which will hold the local source address extracted from the packet

*pDstAddr*

Expects NULL or an `M_BLK_ID` which will hold the local destination address extracted from the packet

*pESrcAddr*

Expects NULL or an `M_BLK_ID` which will hold the end source address extracted from the packet

*pEDstAddr*

Expects NULL or an `M_BLK_ID` which will hold the end destination address extracted from the packet

### VXWORKS AE PROTECTION DOMAINS

Under VxWorks AE, you can call `muxPacketAddrGet()` from within the kernel protection domain only, and the data referenced in the parameters must reside in the kernel protection domain. This restriction does not apply under non-AE versions of VxWorks.

**RETURNS** OK or ERROR.

**ERRNO** S\_muxLib\_NO\_DEVICE

**SEE ALSO** muxLib

---

## muxPacketDataGet()

**NAME** muxPacketDataGet() – return the data from a packet

**SYNOPSIS**

```
STATUS muxPacketDataGet
(
    void *      pCookie,      /* protocol/device binding from muxBind() */
    M_BLK_ID    pMblk,        /* returns the packet data */
    LL_HDR_INFO * pLinkHdrInfo /* returns the packet header information */
)
```

**DESCRIPTION** Any service bound to a driver may use this routine to extract the packet data and remove the link-level header information. This routine copies the header information from the packet referenced in *pMblk* into the LL\_HDR\_INFO structure referenced in *pLinkHdrInfo*.

*pCookie*

Expects the cookie returned from the **muxBind()** call. This cookie indicates the device to which the MUX bound this service.

*pMblk*

Expects a pointer to an **mBlk** or **mBlk** cluster representing a packet containing the data to be returned

*pLinkHdrInfo*

Expects a pointer to an LL\_HDR\_INFO structure into which the packet header information is copied from the incoming **mBlk**

### VXWORKS AE PROTECTION DOMAINS

Under VxWorks AE, you can call **muxPacketDataGet()** from within the kernel protection domain only, and the data referenced in the parameters must reside in the kernel protection domain. This restriction does not apply under non-AE versions of VxWorks.

**RETURNS** OK; or ERROR, if the device type is not recognized.

**ERRNO** S\_muxLib\_NO\_DEVICE

**SEE ALSO** muxLib

---

## muxPollDevAdd()

**NAME** `muxPollDevAdd()` – adds a device to list polled by `tMuxPollTask`

**SYNOPSIS**

```
STATUS muxPollDevAdd
(
    int    unit,           /* Device unit number */
    char * pName          /* Device name */
)
```

**DESCRIPTION** This routine adds a device to list of devices polled by `tMuxPollTask`. It assumes that you have already called `muxPollStart()` and that `tMuxPollTask` is still running.

---

**NOTE:** You cannot use a device for `WDB_COMM_END` type debugging while that device is on the `tMuxPollTask` poll list.

---

**RETURNS** `OK` or `ERROR`

**SEE ALSO** `muxLib`

**M**

---

## muxPollDevDel()

**NAME** `muxPollDevDel()` – removes a device from the list polled by `tMuxPollTask`

**SYNOPSIS**

```
STATUS muxPollDevDel
(
    int    unit,           /* Device unit number */
    char * pName          /* Device name */
)
```

**DESCRIPTION** This routine removes a device from the list of devices polled by `tMuxPollTask`. If you remove the last device on the list, a call to `muxPollDevDel()` also makes an internal call to `muxPollEnd()`. This shuts down `tMuxPollTask` completely.

**RETURNS** `OK` or `ERROR`

**SEE ALSO** `muxLib`

## **muxPollDevStat()**

<b>NAME</b>	<b>muxPollDevStat()</b> – reports whether device is on list polled by <b>tMuxPollTask</b>
<b>SYNOPSIS</b>	<pre>BOOL muxPollDevStat (     int    unit,           /* Device unit number */     char * pName          /* Device name */ )</pre>
<b>DESCRIPTION</b>	This routine returns true or false depending on whether the specified device is on the list of devices polled by <b>tMuxPollTask</b> .
<b>RETURNS</b>	TRUE, if it is; or FALSE.
<b>SEE ALSO</b>	<b>muxLib</b>

---

## **muxPollEnd()**

<b>NAME</b>	<b>muxPollEnd()</b> – shuts down <b>tMuxPollTask</b> and returns devices to interrupt mode
<b>SYNOPSIS</b>	<pre>STATUS muxPollEnd ()</pre>
<b>DESCRIPTION</b>	This routine shuts down <b>tMuxPollTask</b> and returns network devices to run in their interrupt-driven mode.
<b>RETURNS</b>	OK or ERROR
<b>SEE ALSO</b>	<b>muxLib</b>

---

## muxPollReceive()

**NAME** `muxPollReceive()` – now deprecated, see `muxTkPollReceive()`

**SYNOPSIS**

```
STATUS muxPollReceive
(
    void * pCookie,          /* binding instance from muxBind() */
    M_BLK_ID pNBuf          /* a vector of buffers passed to us */
)
```

---

**DESCRIPTION** **NOTE:** This routine has been deprecated in favor of `muxTkPollReceive()`

---

Upper layers can call this routine to poll for a packet.

`pCookie`

Expects the cookie that was returned from `muxBind()`. This cookie indicates which driver to query for available data.

`pNBuf`

Expects a pointer to a buffer chain into which to receive data.

**VXWORKS AE PROTECTION DOMAINS**

Under VxWorks AE, you can call `muxPollReceive()` from within the kernel protection domain only, and the data referenced in the `pCookie` and `pNBuf` parameters must reside in the kernel protection domain. This restriction does not apply under non-AE versions of VxWorks.

**RETURNS** `OK`; `ENETDOWN`, if the `pCookie` argument does not represent a loaded driver; or an error value returned from the driver's registered `endPollReceive()` function.

**ERRNO** `S_muxLib_NO_DEVICE`

**SEE ALSO** `muxLib`

## muxPollSend()

<b>NAME</b>	<b>muxPollSend()</b> – now <b>deprecated</b> , see <b>muxTkPollSend()</b>
<b>SYNOPSIS</b>	<pre>STATUS muxPollSend (     void *   pCookie,           /* binding instance from muxBind() */     M_BLK_ID pNBuf             /* data to be sent */ )</pre>
<b>DESCRIPTION</b>	This routine transmits a packet for the service specified by <i>pCookie</i> . You got this cookie from a previous bind call that bound the service to a particular interface. This <b>muxPollSend()</b> call uses this bound interface to transmit the packet. The <i>pNBuf</i> argument is a buffer ( <b>mBlk</b> ) chain that contains the packet to be sent.
<b>RETURNS</b>	<b>OK</b> ; <b>ENETDOWN</b> , if <i>pCookie</i> does not represent a valid device; <b>ERROR</b> , if the device type is not recognized; or an error value from the device's registered <b>endPollSend()</b> routine.
<b>ERRNO</b>	<b>S_muxLib_NO_DEVICE</b>
<b>SEE ALSO</b>	<b>muxLib</b>

---

## muxPollStart()

**NAME** `muxPollStart()` – initialize and start the MUX poll task

**SYNOPSIS**

```
STATUS muxPollStart
(
    int numDev,           /* Maximum number of devices to support */
                        /* poll mode. */
    int priority,        /* tMuxPollTask priority, not to exceed */
                        /* tNetTask. */
    int delay            /* Delay, in ticks, at end of each polling */
                        /* cycle. */
)
```

**DESCRIPTION**

This routine initializes and starts the MUX poll task, `tMuxPollTask`. This task runs an infinite loop in which it polls each of the interfaces referenced on a list of network interfaces. To add or remove devices from this list, use `muxPollDevAdd()` and `muxPollDevDel()`. Removing all devices from the list automatically triggers a call to `muxPollEnd()`, which shuts down `tMuxPollTask`.

Using the *priority* parameter, you assign the priority to `tMuxPollTask`. Valid values are between 0 and 255, inclusive. However, you must not set the priority of `tMuxPollTask` to exceed that of `tNetTask`. Otherwise, you risk shutting `tNetTask` out from getting processor time. To reset the `tMuxPollTask` priority after launch, use `muxTaskPrioritySet()`.

Using the *delay* parameter, you can set up a delay at the end of each trip through the poll list. To reset the value of this delay after the launch of `tNetTask`, call `muxTaskDelaySet()`.

To shut down `tMuxPollTask`, call `muxPollEnd()`.

**RETURNS** OK or ERROR

**SEE ALSO** `muxLib`

## muxSend()

<b>NAME</b>	<b>muxSend()</b> – send a packet out on a network interface
<b>SYNOPSIS</b>	<pre><b>STATUS</b> muxSend (     void *   pCookie,           /* protocol/device binding from muxBind() */     M_BLK_ID pNBuf             /* data to be sent */ )</pre>
<b>DESCRIPTION</b>	<p>This routine transmits a packet for the service specified by <i>pCookie</i>. You got this cookie from a previous bind call that bound the service to a particular interface. This <b>muxSend()</b> call uses this bound interface to transmit the packet.</p> <p><i>pCookie</i> Expects the cookie returned from <b>muxBind()</b>. This cookie identifies a particular service-to-interface binding.</p> <p><i>pNBuf</i> Expects a pointer to the buffer that contains the packet you want to transmit. Before you call <b>muxSend()</b>, you need to put the addressing information at the head of the buffer. To do this, call <b>muxAddressForm()</b>.</p>
<b>VXWORKS AE PROTECTION DOMAINS</b>	<p>Under VxWorks AE, you can call <b>muxSend()</b> from within the kernel protection domain only, and the data referenced in the <i>pCookie</i> and <i>pNBuf</i> parameters must reside in the kernel protection domain. This restriction does not apply under non-AE versions of VxWorks.</p>
<b>RETURNS</b>	<b>OK</b> ; <b>ENETDOWN</b> , if <i>pCookie</i> does not represent a valid binding; or <b>ERROR</b> , if the driver's <b>endSend()</b> routine fails.
<b>ERRNO</b>	<b>S_muxLib_NO_DEVICE</b>
<b>SEE ALSO</b>	<b>muxLib</b>

---

## **muxShow()**

<b>NAME</b>	<b>muxShow()</b> – display configuration of devices registered with the MUX
<b>SYNOPSIS</b>	<pre>void muxShow (     char * pDevName,          /* pointer to device name, or NULL for all */     int    unit              /* unit number for a single device */ )</pre>
<b>DESCRIPTION</b>	<p>If the <i>pDevName</i> and <i>unit</i> arguments specify an existing device, this routine reports the name and type of each protocol bound to it. Otherwise, if <i>pDevName</i> is NULL, the routine displays the entire list of existing devices and their associated protocols.</p> <p><i>pDevName</i> A string that contains the name of the device, or a null pointer to indicate “all devices.”</p> <p><i>unit</i> Specifies the unit number of the device (if <i>pDevName</i> is not a null pointer).</p>
<b>RETURNS</b>	N/A
<b>SEE ALSO</b>	<b>muxLib</b>

---

## **muxTaskDelayGet()**

<b>NAME</b>	<b>muxTaskDelayGet()</b> – get the delay on the polling task
<b>SYNOPSIS</b>	<pre>STATUS muxTaskDelayGet (     int* pDelay )</pre>
<b>DESCRIPTION</b>	<p>This routine returns the amount of delay (in ticks) that is inserted between the polling runs of <b>tMuxPollTask</b>. This value is written to the location specified by <i>pDelay</i>.</p>
<b>RETURNS</b>	OK; or <b>ERROR</b> , if NULL is passed in the <i>pDelay</i> variable.
<b>SEE ALSO</b>	<b>muxLib</b>

## **muxTaskDelaySet()**

**NAME** `muxTaskDelaySet()` – set the inter-cycle delay on the polling task

**SYNOPSIS** `STATUS muxTaskDelaySet`  
(  
    **int** *delay*  
)

**DESCRIPTION** This routine sets up a delay (measured in ticks) that is inserted at the end of each run through the list of devices polled by **tMuxPollTask**.

**RETURNS** **OK**; or **ERROR**, if you specify a delay less than zero.

**SEE ALSO** `muxLib`

---

## **muxTaskPriorityGet()**

**NAME** `muxTaskPriorityGet()` – get the priority of **tMuxPollTask**

**SYNOPSIS** `STATUS muxTaskPriorityGet`  
(  
    **int\*** *pPriority*  
)

**DESCRIPTION** This routine returns the current priority of **tMuxPollTask**. This value is returned to the location specified by the *pPriority* parameter.

**RETURNS** **OK**; or **ERROR**, if **NULL** is passed in the *pPriority* parameter.

**SEE ALSO** `muxLib`

---

## muxTaskPrioritySet( )

<b>NAME</b>	<code>muxTaskPrioritySet( )</code> – reset the priority of <code>tMuxPollTask</code>
<b>SYNOPSIS</b>	<pre> STATUS muxTaskPrioritySet (     int priority ) </pre>
<b>DESCRIPTION</b>	This routine resets the priority of a running <code>tMuxPollTask</code> . Valid task priorities are values between zero and 255 inclusive. However, do not set the priority of <code>tMuxPollTask</code> to exceed that of <code>tNetTask</code> . Otherwise, you will shut out <code>tNetTask</code> from getting any processor time.
<b>RETURNS</b>	OK; or <code>ERROR</code> , if you specify a non-valid priority value.
<b>SEE ALSO</b>	<code>muxLib</code>

---

## muxTkBind( )

<b>NAME</b>	<code>muxTkBind( )</code> – bind an NPT protocol to a driver
<b>SYNOPSIS</b>	<pre> void * muxTkBind (     char * pName,           /* interface name, for example, ln, ei,... */     int  unit,             /* unit number */     BOOL (* stackRcvRtn) (void* ,long, M_BLK_ID, void * ),                           /* receive function to be called. */     STATUS (* stackShutdownRtn) (void * ),                           /* routine to call to shutdown the stack */     STATUS (* stackTxRestartRtn) (void * ),                           /* routine to tell the stack it can transmit */     void (* stackErrorRtn) (void* , END_ERR* ),                           /* routine to call on an error. */     long  type,            /* protocol type from RFC1700 and many */                           /* other sources (for example, 0x800 is IP) */     char * pProtoName,     /* string name for protocol */     void * pNetCallbackId, /* returned to network service sublayer */                           /* during recv */     void * pNetSvcInfo,    /* reference to netSrvInfo structure */     void * pNetDrvInfo     /* reference to netDrvInfo structure */ ) </pre>

**DESCRIPTION**

A network protocol, network service, or service sublayer uses this routine to bind to a specific driver. This bind routine is valid both for END and NPT drivers, but the specified stack routine parameters must use the NPT function prototypes, and are somewhat different from those used with ***muxBind()***.

The driver is specified by the *pName* and *unit* arguments, (for example, ln and 0, ln and 1, or ei and 0).

*pName*

Expects a pointer to a character string that contains the name of the device that this network service wants to use to send and receive packets.

*unit*

Expects the unit number of the device of the type indicated by *pName*.

*stackRcvRtn*

Expects a pointer to the function that the MUX will call when it wants to pass a packet up to the network service. For a description of how to write this routine, see the *WindNet TCP/IP Network Programmer's Guide*

*stackShutdownRtn*

Expects a pointer to the function that the MUX will call to shutdown the network service. For a description of how to write such a routine, see the *WindNet TCP/IP Network Programmer's Guide*

*stackTxRestartRtn*

Expects a pointer to the function that the MUX will call after packet transmission has been suspended, to tell the network service that it can continue transmitting packets. For a description of how to write this routine, see the *WindNet TCP/IP Network Programmer's Guide*

*stackErrorRtn*

Expects a pointer to the function that the MUX will call to give errors to the network service. For a description of how to write this routine, see the section *WindNet TCP/IP Network Programmer's Guide*

*type*

Expects a value that indicates the protocol type. The MUX uses this type to prioritize a network service as well as to modify its capabilities. For example, a network service of type ***MUX\_PROTO\_SNARF*** has the highest priority (see the description of protocol prioritizing provided in *WindNet TCP/IP Network Programmer's Guide*. Aside from ***MUX\_PROTO\_SNARF*** and ***MUX\_PROTO\_PROMISC***, valid network service types include any of the values specified in RFC 1700, or can be user-defined.

The *stackRcvRtn* is called whenever the MUX has a packet of the specified type. If the type is ***MUX\_PROTO\_PROMISC***, the protocol is considered promiscuous and will get all of the packets that have not been consumed by any other protocol. If the type is ***MUX\_PROTO\_SNARF***, it will get all of the packets that the MUX sees.

If the type is ***MUX\_PROTO\_OUTPUT***, this network service is an output protocol and all packets that are to be output on this device are first passed to *stackRcvRtn* routine rather

than being sent to the device. This can be used by a network service that needs to send packets directly to another network service, or in a loop-back test. If the *stackRcvRtn* returns **OK**, the packet is consumed and as no longer available. The *stackRcvRtn* for an output protocol may return **ERROR** to indicate that it wants to look at the packet without consuming it.

*pProtoName*

Expects a pointer to a character string for the name of this network service. This string can be **NULL**, in which case a network service name is assigned internally.

*pNetCallbackId*

Expects a pointer to a structure defined by the protocol. This argument is passed up to the protocol as the first argument of all the callbacks. This argument corresponds to the *pSpare* argument in **muxBind()**

*pNetSvcInfo*

Reference to an optional structure specifying network service layer information needed by the driver

*pNetDrvInfo*

Reference to an optional structure specifying network driver information needed by the network protocol, network service, or service sublayer

- RETURNS** A cookie that uniquely represents the binding instance, or **NULL** if the bind fails.
- ERRNO** **S\_muxLib\_NO\_DEVICE**, **S\_muxLib\_END\_BIND\_FAILED**, **S\_muxLib\_NO\_TK\_DEVICE**, **S\_muxLib\_NOT\_A\_TK\_DEVICE**, **S\_muxLib\_ALREADY\_BOUND**, **S\_muxLib\_ALLOC\_FAILED**
- SEE ALSO** **muxTkLib**, **muxBind()**

---

## **muxTkCookieGet()**

- NAME** **muxTkCookieGet()** – returns the cookie for a device
- SYNOPSIS**
- ```
void *muxTkCookieGet
(
    char * pName,           /* Device Name */
    int   unit             /* Device Unit */
)
```
- DESCRIPTION** This routine returns the cookie for a device.
- RETURNS** a cookie to the device or **NULL** if unsuccessful
- SEE ALSO** **muxTkLib**

---

## **muxTkDrvCheck()**

- NAME** `muxTkDrvCheck()` – checks if the device is an NPT or an END interface
- SYNOPSIS**
- ```
int muxTkDrvCheck
(
    char * pDevName          /* device name */
)
```
- DESCRIPTION** This function returns 1 if the driver indicated by *pDevName* is of the Toolkit (NPT) paradigm, and 0 (zero) if it is an END. This routine is called by the network service sublayer so that it can discover the driver type before it binds to it via the MUX.
- RETURNS** 1 for an NPT driver, 0 for an END or other driver, or **ERROR** (-1) if no device is found with the given name
- SEE ALSO** `muxTkLib`, `muxTkBind()`, `muxBind()`

---

## **muxTkPollReceive()**

- NAME** `muxTkPollReceive()` – poll for a packet from a NPT or END driver
- SYNOPSIS**
- ```
STATUS muxTkPollReceive
(
    void * pCookie,          /* cookie from muxTkBind routine */
    M_BLK_ID pNBuf,         /* a vector of buffers passed to us */
    void * pSpare            /* a reference to spare data is returned here */
)
```
- DESCRIPTION** This is the routine that an upper layer can call to poll for a packet. Any service type retrieved from the MAC frame is passed via the reserved member of the `M_BLK` header. This API effectively replaces `muxPollReceive()` for both END and NPT drivers. For an NPT driver its `pollReceive()` entry point is called based on the new prototype:
- ```
STATUS nptPollReceive
(
    END_OBJ * pEND,          /* END object */
    M_BLK_ID pPkt,          /* network packet buffer */
    long * pNetSvc,         /* service type from MAC frame */
    long * pNetOffset,      /* offset to network packet */
)
```

```
void *    pSpareData    /* optional network service data */
)
```

The `pollReceive()` entry point for an END driver uses the original prototype:

```
STATUS endPollRcv
(
  END_OBJ * pEND,    /* END object */
  M_BLK_ID pPkt,    /* network packet buffer */
)
```

An END driver must continue to provide the `packetDataGet()` entry point

*pCookie*

Expects the cookie that was returned from `muxBind()` or `muxTkBind()`. This “cookie” identifies the driver.

*pNBuf*

Expects a pointer to a buffer chain into which incoming data will be put.

*pSpareData*

A pointer to any optional spare data provided by a NPT driver. Always NULL with an END driver.

<b>RETURNS</b>	OK; EAGAIN, if no packet was available; ENETDOWN, if the <code>pCookie</code> does not represent a loaded driver; or an error value returned from the driver’s registered <code>pollReceive()</code> function.
<b>ERRNO</b>	S_muxLib_NO_DEVICE
<b>SEE ALSO</b>	<code>muxTkLib</code>

---

## muxTkPollSend()

**NAME** `muxTkPollSend()` – send a packet out in polled mode to an END or NPT interface

**SYNOPSIS**

```
STATUS muxTkPollSend
(
  void *    pCookie,          /* returned by muxTkBind() */
  M_BLK_ID pNBuf,           /* data to be sent */
  char *    dstMacAddr,     /* destination MAC address */
  USHORT   netType,         /* network protocol that is calling us * is */
                                /* netType redundant? * */
  void *    pSpareData      /* spare data passed on each send */
)
```

***muxTkPollSend()*****DESCRIPTION**

This routine uses *pCookie* to find a specific network interface and use that driver's **pollSend()** routine to transmit a packet.

This routine replaces the **muxPollSend()** routine for both END and NPT drivers.

When using this routine, the driver does not need to call **muxAddressForm()** to complete the packet, nor does it need to prepend an **mBlk** of type **ME\_IFADDR** containing the destination address.

An NPT driver's **pollSend()** entry point is called based on this prototype:

```
STATUS nptPollSend
(
    END_OBJ * pEND,          /* END object */
    M_BLK_ID pPkt,          /* network packet to transmit */
    char *   pDstAddr,      /* destination MAC address */
    long    netType         /* network service type */
    void *   pSpareData     /* optional network service data */
)
```

The **pollSend()** entry point for an END uses this prototype:

```
STATUS endPollSend
(
    END_OBJ * pEND,        /* END object */
    M_BLK_ID pPkt,        /* network packet to transmit */
)
```

An END driver must provide the **addressForm()** entry point to construct the appropriate link-level header. The *pDst* and *pSrc* **M\_BLK** arguments to that routine supply the link-level addresses with the **mData** and **mLen** fields. The reserved field of the destination **M\_BLK** contains the network service type. Both arguments *must* be treated as read-only.

*pCookie*

Expects the cookie returned from **muxBind()** or **muxTkBind()**. This cookie identifies the device to which the MUX has bound this protocol.

*pNBuf*

The network packet to be sent.

*dstMacAddr*

Destination MAC address to which packet is to be sent

*netType*

Network service type that will be used to identify the payload data in the MAC frame.

*pSpareData*

Reference to any additional data the network service wants to pass to the driver during the send operation.

<b>RETURNS</b>	OK, ENETDOWN if <i>pCookie</i> doesn't represent a valid device, or an error if the driver's <code>pollSend()</code> routine fails.
<b>ERRNO</b>	S_muxLib_NO_DEVICE
<b>SEE ALSO</b>	<code>muxTkLib</code>

---

## **muxTkReceive()**

**NAME** `muxTkReceive()` – receive a packet from a NPT driver

**SYNOPSIS**

```

STATUS muxTkReceive
(
    void *   pCookie,           /* cookie passed in endLoad() call */
    M_BLK_ID pMblk,           /* a buffer passed to us. */
    long    netSvcOffset,     /* offset to network datagram in the packet */
    long    netSvcType,       /* network service type */
    BOOL    uniPromiscuous,   /* TRUE when driver is in promiscuous mode */
    void *   pSpareData       /* out of band data */
)

```

**DESCRIPTION** This is the routine that the NPT driver calls to hand a packet to the MUX. This routine forwards the received `mBlk` chain to the network service sublayer by calling its registered `stackRcvRtn()`.

Typically, a driver includes an interrupt handling routine to process received packets. It should keep processing to a minimum during interrupt context and then arrange for processing of the received packet within task context.

Once the frame has been validated, the driver should pass it to the MUX with the `receiveRtn` member of its `END_OBJ` structure. This routine has the same prototype as (and typically is) `muxTkReceive()`.

Depending on the protocol type (for example, `MUX_PROTO_SNARF` or `MUX_PROTO_PROMISC`), this routine either forwards the received packet chain unmodified or it changes the data pointer in the `mBlk` to strip off the frame header before forwarding the packet.

*pCookie*

Expects the `END_OBJ` pointer returned by the driver's `endLoad()` or `nptLoad()` function

*pMblk*

Expects a pointer to the `mBlk` structure containing the packet that has been received

*netSvcOffset*

Expects an offset into the frame to the point where the data field (the network service layer header) begins

*netSvcType*

Expects the network service type of the service for which the packet is destined (typically this value can be found in the header of the received frame)

*uniPromiscuous*

Expects a boolean set to **TRUE** when driver is in promiscuous mode and receives a unicast or a multicast packet not intended for this device. When **TRUE** the packet is not handed over to network services other than those registered as SNARF or PROMISCUOUS.

*pSpareData*

Expects a pointer to any spare data the driver needs to pass up to the network service layer, or **NULL**

**RETURNS** OK or **ERROR**.  
**ERRNO** S\_muxLib\_NO\_DEVICE  
**SEE ALSO** muxTkLib

---

## **muxTkSend()**

**NAME** muxTkSend() – send a packet out on a Toolkit or END network interface

**SYNOPSIS**

```
STATUS muxTkSend
(
    void *   pCookie,           /* returned by muxTkBind() */
    M_BLK_ID pNBuf,           /* data to be sent */
    char *   dstMacAddr,      /* destination MAC address */
    USHORT  netType,          /* network protocol that is calling us * is */
                                /* netType redundant? * */
    void *   pSpareData       /* spare data passed on each send */
)
```

**DESCRIPTION** This routine uses *pCookie* to find a specific network interface and uses that driver's send routine to transmit a packet.

The transmit entry point for an NPT driver uses the following prototype:

```

STATUS nptSend
(
    END_OBJ * pEND,          /* END object */
    M_BLK_ID pPkt,          /* network packet to transmit */
    char *   pDstAddr,      /* destination MAC address */
    int      netType        /* network service type */
    void *   pSpareData     /* optional network service data */
)

```

The transmit entry point for an END driver the following prototype:

```

STATUS endSend
(
    void *   pEND,          /* END object */
    M_BLK_ID pPkt,          /* Network packet to transmit */
)

```

An END driver must continue to provide the **addressForm()** entry point to construct the appropriate link-level header. The *pDst* and *pSrc* **M\_BLK** arguments to that routine supply the link-level addresses with the **mData** and **mLen** fields. The reserved field of the destination **M\_BLK** contains the network service type. Both arguments *must* be treated as read-only.

To send a fully formed physical layer frame to a device using an NPT driver (which typically forms the frame itself), set the **M\_L2HDR** flag in the **mBlk** header.

A driver may be written so that it returns the error **END\_ERR\_BLOCK** if the driver has insufficient resources to transmit data. The network service sublayer can use this feedback to establish a flow control mechanism by holding off on making any further calls to **muxTkSend()** until the device is ready to restart transmission, at which time the device should call **muxTxRestart()** which will call the service sublayer's **stackRestartRtn()** that was registered for the interface at bind time.

*pCookie*

Expects the cookie returned from **muxTkBind()**. This Cookie identifies the device to which the MUX has bound this protocol.

*pNBuf*

The network packet to be sent, formed into an **mBlk** chain.

*dstMacAddr*

Destination MAC address to which packet is to be sent, determined perhaps by calling the address resolution function that was registered for this service/device interface.

*netType*

Network service type of the sending service. This will be used to identify the payload type in the MAC frame.

**M**

*pSpareData*

Reference to any additional data the network service wants to pass to the driver during the send operation.

---

**NOTE:** A driver may return `END_ERR_BLOCK` if it is temporarily unable to complete the send, and then call `muxTxRestart()` to indicate that it is again able to send data. If the driver has been written in this way, `muxTkSend()` will pass the `ERR_END_BLOCK` back as its own return value and the service can wait for its `stackRestartRtn()` callback routine to be called before trying the send operation again.

---

**RETURNS** OK; `ENETDOWN`, if *pCookie* doesn't represent a valid device; or an error, if the driver's `send()` routine fails.

**ERRNO** `S_muxLib_NO_DEVICE`

**SEE ALSO** `muxTkLib`

---

## **muxUnbind()**

**NAME** `muxUnbind()` – detach a network service from the specified device

**SYNOPSIS**

```
STATUS muxUnbind
(
    void * pCookie,          /* binding instance from muxBind() or */
                           /* muxTkBind() */
    long   type,            /* type passed to muxBind() or muxTkBind() call */
    FUNCPTR stackRcvRtn    /* pointer to stack receive routine */
)
```

**DESCRIPTION** This routine disconnects a network service from the specified device. The *pCookie* argument indicates the service/device binding returned by the `muxBind()` or `muxTkBind()` routine. The *type* and *stackRcvRtn* arguments must also match the values given to the original `muxBind()` or `muxTkBind()` call.

---

**NOTE:** If `muxUnbind()` returns `ERROR`, and `errno` is set to `EINVAL`, this indicates that the device is not bound to the service.

---

**RETURNS** OK; or `ERROR`, if `muxUnbind()` fails.

**VXWORKS AE PROTECTION DOMAINS**

Under VxWorks AE, you can call `muxUnBind()` from within the kernel protection

domain only, and the data referenced in the *stackRcvRtn* and *pCookie* parameters must reside in the kernel protection domain. This restriction does not apply under non-AE versions of VxWorks.

**ERRNO** EINVAL, S\_muxLib\_NO\_DEVICE

**SEE ALSO** muxLib

---

## mv()

**NAME** mv() – mv file into other directory.

**SYNOPSIS**

```

STATUS mv
(
    const char * src,          /* source file name or wildcard */
    const char * dest         /* destination name or directory */
)

```

**DESCRIPTION** This function is similar to **rename()** but behaves somewhat more like the UNIX program “mv”, it will overwrite files.

This command moves the *src* file or directory into a file which name is passed in the *dest* argument, if *dest* is a regular file or does not exist. If *dest* name is a directory, the source object is moved into this directory as with the same name, if *dest* is NULL, the current directory is assumed as the destination directory. *src* may be a single file name or a path containing a wildcard pattern, in which case all files or directories matching the pattern will be moved to *dest* which must be a directory in this case.

**EXAMPLES**

```

-> mv( "/sd0/dir1", "/sd0/dir2")
-> mv( "/sd0/*.tmp", "/sd0/junkdir")
-> mv( "/sd0/FILE1.DAT", "/sd0/dir2/f001.dat")

```

**RETURNS** OK, or ERROR if any of the files or directories could not be moved, or if *src* is a pattern but the destination is not a directory.

**SEE ALSO** usrFsLib

## nanosleep()

**NAME** nanosleep() – suspend the current task until the time interval elapses (POSIX)

**SYNOPSIS**

```
int nanosleep
(
    const struct timespec * rqt, /* time to delay */
    struct timespec *      rmt /* premature wakeup (NULL=no result) */
)
```

**DESCRIPTION** This routine suspends the current task for a specified time *rqt* or until a signal or event notification is made.

The suspension may be longer than requested due to the rounding up of the request to the timer's resolution or to other scheduling activities (*e.g.*, a higher priority task intervenes).

The **timespec** structure is defined as follows:

```
struct timespec
{
    time_t tv_sec;          /* interval = tv_sec*10**9 + tv_nsec */
    long tv_nsec;          /* seconds */
};                          /* nanoseconds (0 - 1,000,000,000) */
```

If *rmt* is non-NULL, the **timespec** structure is updated to contain the amount of time remaining. If *rmt* is NULL, the remaining time is not returned. The *rqt* parameter is greater than 0 or less than or equal to 1,000,000,000.

**RETURNS** 0 (OK), or -1 (ERROR) if the routine is interrupted by a signal or an asynchronous event notification, or *rqt* is invalid.

**ERRNO** EINVAL, EINTR

**SEE ALSO** timerLib, sleep(), taskDelay()

---

## netBufLibInit()

<b>NAME</b>	<b>netBufLibInit()</b> – initialize <b>netBufLib</b>
<b>SYNOPSIS</b>	<b>STATUS netBufLibInit (void)</b>
<b>DESCRIPTION</b>	This routine executes during system startup if <b>INCLUDE_NETWORK</b> is defined when the image is built. It links the network buffer library into the image.
<b>RETURNS</b>	OK or ERROR.
<b>SEE ALSO</b>	<b>netBufLib</b>

---

## netCIBlkFree()

<b>NAME</b>	<b>netCIBlkFree()</b> – free a <b>cIBlk</b> -cluster construct back to the memory pool
<b>SYNOPSIS</b>	<pre>void netCIBlkFree (     NET_POOL_ID pNetPool,    /* pointer to the net pool */     CL_BLK_ID   pCIBlk      /* pointer to the cIBlk to free */ )</pre>
<b>DESCRIPTION</b>	This routine decrements the reference counter in the specified <b>cIBlk</b> . If the reference count falls to zero, this routine frees both the <b>cIBlk</b> and its associated cluster back to the specified memory pool.
<b>VXWORKS AE PROTECTION DOMAINS</b>	Under VxWorks AE, you can call this function from within the kernel protection domain only. In addition, all arguments to this function can reference only that data which is valid in the kernel protection domain. This restriction does not apply under non-AE versions of VxWorks.
<b>RETURNS</b>	N/A
<b>SEE ALSO</b>	<b>netBufLib</b>

---

## netCIBlkGet()

**NAME** netCIBlkGet() – get a cIBlk

**SYNOPSIS**

```
CL_BLK_ID netCIBlkGet
(
    NET_POOL_ID pNetPool,    /* pointer to the net pool */
    int         canWait      /* M_WAIT/M_DONTWAIT */
)
```

**DESCRIPTION** This routine gets a cIBlk from the specified memory pool.

*pNetPool*

Expects a pointer to the pool from which you want a cIBlk.

*canWait*

Expects either M\_WAIT or M\_DONTWAIT. If no cIBlk is immediately available, the M\_WAIT value allows this routine to repeat the allocation attempt after performing garbage collection. It omits these steps when the M\_DONTWAIT value is used.

### VXWORKS AE PROTECTION DOMAINS

Under VxWorks AE, you can call this function from within the kernel protection domain only. In addition, all arguments to this function can reference only that data which is valid in the kernel protection domain. Likewise, the returned ID is valid in the kernel protection domain only. This restriction does not apply under non-AE versions of VxWorks.

**RETURNS** CL\_BLK\_ID or a NULL if no cIBlk was available.

**SEE ALSO** netBufLib

---

## netCIBlkJoin()

**NAME** netCIBlkJoin() – join a cluster to a cIBlk structure

**SYNOPSIS**

```
CL_BLK_ID netCIBlkJoin
(
    CL_BLK_ID pCIBlk,        /* pointer to a cluster Blk */
    char *    pCIBuf,        /* pointer to a cluster buffer */
    int       size,          /* size of the cluster buffer */
)
```

```

FUNCPTR  pFreeRtn,      /* pointer to the free routine */
int      arg1,          /* argument 1 of the free routine */
int      arg2,          /* argument 2 of the free routine */
int      arg3           /* argument 3 of the free routine */
)

```

**DESCRIPTION** This routine joins the previously reserved cluster specified by *pClBuf* to the previously reserved **clBlk** structure specified by *pClBlk*. The *size* parameter passes in the size of the cluster referenced in *pClBuf*. The arguments *pFreeRtn*, *arg1*, *arg2*, *arg3* set the values of the **pCLFreeRtn**, **clFreeArg1**, **clFreeArg2**, and **clFreeArg1**, members of the specified **clBlk** structure.

#### VXWORKS AE PROTECTION DOMAINS

Under VxWorks AE, you can call this function from within the kernel protection domain only. In addition, all arguments to this function can reference only that data which is valid in the kernel protection domain. Likewise, the returned ID is valid in the kernel protection domain only. This restriction does not apply under non-AE versions of VxWorks.

**RETURNS** **CL\_BLK\_ID** or **NULL**.

**SEE ALSO** **netBufLib**

N

---

## netClFree()

**NAME** **netClFree()** – free a cluster back to the memory pool

**SYNOPSIS**

```

void netClFree
(
    NET_POOL_ID pNetPool,      /* pointer to the net pool */
    UCHAR *     pClBuf         /* pointer to the cluster buffer */
)

```

**DESCRIPTION** This routine returns the specified cluster buffer back to the specified memory pool.

#### VXWORKS AE PROTECTION DOMAINS

Under VxWorks AE, you can call this function from within the kernel protection domain only. In addition, all arguments to this function can reference only that data which is valid in the kernel protection domain. This restriction does not apply under non-AE versions of VxWorks.

**RETURNS** N/A

**SEE ALSO** netBufLib

---

## netClPoolIdGet()

**NAME** netClPoolIdGet() – return a CL\_POOL\_ID for a specified buffer size

**SYNOPSIS**

```
CL_POOL_ID netClPoolIdGet
(
    NET_POOL_ID pNetPool,    /* pointer to the net pool */
    int         bufSize,     /* size of the buffer */
    BOOL        bestFit      /* TRUE/FALSE */
)
```

**DESCRIPTION** This routine returns a CL\_POOL\_ID for a cluster pool containing clusters that match the specified *bufSize*. If *bestFit* is TRUE, this routine returns a CL\_POOL\_ID for a pool that contains clusters greater than or equal to *bufSize*. If *bestFit* is FALSE, this routine returns a CL\_POOL\_ID for a cluster from whatever cluster pool is available. If the memory pool specified by *pNetPool* contains only one cluster pool, *bestFit* should always be FALSE.

### VXWORKS AE PROTECTION DOMAINS

Under VxWorks AE, you can call this function from within the kernel protection domain only. In addition, all arguments to this function can reference only that data which is valid in the kernel protection domain. Likewise, the returned ID is valid in the kernel protection domain only. This restriction does not apply under non-AE versions of VxWorks.

**RETURNS** CL\_POOL\_ID or NULL.

**SEE ALSO** netBufLib

---

## netClusterGet()

**NAME** `netClusterGet()` – get a cluster from the specified cluster pool

**SYNOPSIS**

```
char * netClusterGet
(
    NET_POOL_ID pNetPool,    /* pointer to the net pool */
    CL_POOL_ID  pClPool     /* ptr to the cluster pool */
)
```

**DESCRIPTION** This routine gets a cluster from the specified cluster pool *pClPool* within the specified memory pool *pNetPool*.

### VXWORKS AE PROTECTION DOMAINS

Under VxWorks AE, you can call this function from within the kernel protection domain only. In addition, all arguments to this function can reference only that data which is valid in the kernel protection domain. Likewise, the returned ID is valid in the kernel protection domain only. This restriction does not apply under non-AE versions of VxWorks.

**RETURNS** This routine returns a character pointer to a cluster buffer or NULL if none was available.

**SEE ALSO** `netBufLib`

---

## netDevCreate()

**NAME** `netDevCreate()` – create a remote file device

**SYNOPSIS**

```
STATUS netDevCreate
(
    char * devName,    /* name of device to create */
    char * host,      /* host this device will talk to */
    int  protocol     /* remote file access protocol 0 = RSH, 1 = FTP */
)
```

**DESCRIPTION** This routine creates a remote file device. Normally, a network device is created for each remote machine whose files are to be accessed. By convention, a network device name is the remote machine name followed by a colon ":". For example, for a UNIX host on the

network whose name is “wrs”, files can be accessed by creating a device called “wrs:”. Files can be accessed via RSH as follows:

```
netDevCreate ("wrs:", "wrs", rsh);
```

The file `/usr/dog` on the UNIX system “wrs” can now be accessed as “wrs:/usr/dog” via RSH.

Before creating a device, the host must have already been created with `hostAdd()`.

**RETURNS** OK or ERROR.

**SEE ALSO** `netDrv`, `hostAdd()`

---

## netDevCreate2()

**NAME** `netDevCreate2()` – create a remote file device with fixed buffer size

**SYNOPSIS**

```
STATUS netDevCreate2
(
    char * devName,      /* name of device to create */
    char * host,        /* host this device will talk to */
    int  protocol,      /* remote file access protocol 0 = RSH, 1 = FTP */
    UINT bufSize        /* size of buffer in NET_FD */
)
```

**DESCRIPTION** This routine creates a remote file device, just like `netDevCreate()`, but it allows very large files to be accessed without loading the entire file to memory. The fourth parameter *bufSize* specifies the amount of memory. If *bufSize* is zero, the behavior is exactly the same as `netDevCreate()`. If *bufSize* is not zero, the following restrictions apply:

- `O_RDONLY`, `O_WRONLY` open mode are supported, but not `O_RDWR` open mode.
- seek is supported in `O_RDONLY` open mode, but not in `O_WRONLY` open mode.
- backward seek might be slow if it is beyond the buffer.

**RETURNS** OK or ERROR.

**SEE ALSO** `netDrv`, `netDevCreate()`

---

## netDrv()

- NAME** `netDrv()` – install the network remote file driver
- SYNOPSIS** `STATUS netDrv (void)`
- DESCRIPTION** This routine initializes and installs the network driver. It must be called before other network remote file functions are performed. It is called automatically when `INCLUDE_NET_DRV` is defined.
- VXWORKS AE PROTECTION DOMAINS**  
Under VxWorks AE, you can call this function from within the kernel protection domain only. This restriction does not apply under non-AE versions of VxWorks.
- RETURNS** `OK` or `ERROR`.
- SEE ALSO** `netDrv`

---

## netDrvDebugLevelSet()

- NAME** `netDrvDebugLevelSet()` – set the debug level of the `netDrv` library routines
- SYNOPSIS** `STATUS netDrvDebugLevelSet`  
(  
    **UINT32** `debugLevel`                    /\* `NETDRV_DEBUG_OFF`, `NETDRV_DEBUG_ERRORS`, \*/  
  /\* `NETDRV_DEBUG_ALL` \*/  
)
- DESCRIPTION** This routine enables the debugging of calls to the net driver. The argument `NETLIB_DEBUG_ERRORS` will display only error messages to the console. The argument `NETLIB_DEBUG_ALL` will display warnings and errors to the console.
- RETURNS** `OK`, or `ERROR` if the debug level is invalid
- SEE ALSO** `netDrv`

---

## netDrvFileDoesNotExistInstall()

- NAME** netDrvFileDoesNotExistInstall() – install an applet to test if a file exists
- SYNOPSIS**
- ```
STATUS netDrvFileDoesNotExistInstall
(
    FUNCPTR pAppletteRtn /* function that returns TRUE or FALSE */
)
```
- DESCRIPTION** Install a function to test if a file exists. *pAppletteRtn* should be of the following format:
- ```
STATUS appletteRoutine
(
    char *filename, /* filename queried */
    char *response /* server response string */
)
```
- The **netDrv()** routine calls the applet during an open with **O\_CREAT**. The system performs an **NLST** command and uses the applet to parse the response. The routine compensates for server response implementation variations. The applet should return **OK** if the file is not found and **ERROR** if the file is found.
- RETURNS** **OK**, installation successful; **ERROR**, installation error.
- SEE ALSO** netDrv, open()

---

## netHelp()

- NAME** netHelp() – print a synopsis of network routines
- SYNOPSIS**
- ```
void netHelp (void)
```
- DESCRIPTION** This command prints a brief synopsis of network facilities typically called from the shell.
- ```
hostAdd      "hostname", "inetaddr" - add a host to remote host table;
                                         "inetaddr" must be in standard
                                         Internet address format e.g. "90.0.0.4"
hostShow
netDevCreate "devname", "hostname", protocol
                                         - create an I/O device to access
                                         files on the specified host
                                         (protocol 0=rsh, 1=ftp)
```

```

routeAdd      "destaddr","gateaddr" - add route to route table
routeDelete  "destaddr","gateaddr" - delete route from route table
routeShow
iam          "usr":["passwd"]      - specify the user name by which
                                     you will be known to remote
                                     hosts (and optional password)
whoami
rlogin      "host"                - print the current remote ID
                                     - log in to a remote host;
                                     "host" can be inet address or
                                     host name in remote host table
ifShow      ["ifname"]           - show info about network interfaces
inetstatShow
tcpstatShow
udpstatShow
ipstatShow
icmpstatShow
arptabShow
mbufShow    - show mbuf statistics

```

**RETURNS** N/A

**SEE ALSO** *usrLib, VxWorks Programmer's Guide: Target Shell*

---

## netLibInit()

**NAME** netLibInit() – initialize the network package

**SYNOPSIS** **STATUS** netLibInit (void)

**DESCRIPTION** This creates the network task job queue, and spawns the network task **netTask()**. It should be called once to initialize the network. This is done automatically when **INCLUDE\_NET\_LIB** is defined.

### **VXWORKS AE PROTECTION DOMAINS**

Under VxWorks AE, you can call this function from within the kernel protection domain only. This restriction does not apply under non-AE versions of VxWorks.

**RETURNS** OK, or **ERROR** if network support cannot be initialized.

**SEE ALSO** *netLib, usrConfig, netTask()*

---

## netMblkChainDup()

**NAME** netMblkChainDup() – duplicate an mBlk chain

**SYNOPSIS**

```
M_BLK_ID netMblkChainDup
(
    NET_POOL_ID pNetPool,      /* pointer to the pool */
    M_BLK_ID    pMblk,         /* pointer to source mBlk chain*/
    int         offset,        /* offset to duplicate from */
    int         len,           /* length to copy */
    int         canWait        /* M_DONTWAIT/M_WAIT */
)
```

**DESCRIPTION** This routine makes a copy of an mBlk chain starting at *offset* bytes from the beginning of the chain and continuing for *len* bytes. If *len* is M\_COPYALL, then this routine will copy the entire mBlk chain from the *offset*.

This routine copies the references from a source *pMblk* chain to a newly allocated mBlk chain. This lets the two mBlk chains share the same cBlk-cluster constructs. This routine also increments the reference count in the shared cBlk. The *pMblk* expects a pointer to the source mBlk chain. The *pNetPool* parameter expects a pointer to the netPool from which the new mBlk chain is allocated.

The *canWait* parameter determines the behavior if any required mBlk is not immediately available. A value of M\_WAIT allows this routine to repeat the allocation attempt after performing garbage collection. The M\_DONTWAIT value prevents those extra steps.

### VXWORKS AE PROTECTION DOMAINS

Under VxWorks AE, you can call this function from within the kernel protection domain only. In addition, all arguments to this function can reference only that data which is valid in the kernel protection domain. Likewise, the returned ID is valid in the kernel protection domain only. This restriction does not apply under non-AE versions of VxWorks.

**SEE ALSO** netBufLib, netMblkDup()

**RETURNS** A pointer to the newly allocated mBlk chain or NULL.

**ERRNO** S\_netBufLib\_INVALID\_ARGUMENT  
S\_netBufLib\_NO\_POOL\_MEMORY

---

## netMblkClChainFree()

<b>NAME</b>	<code>netMblkClChainFree()</code> – free a chain of <b>mBlk-clBlk</b> -cluster constructs
<b>SYNOPSIS</b>	<pre>void netMblkClChainFree (     M_BLK_ID pMblk          /* pointer to the mBlk */ )</pre>
<b>DESCRIPTION</b>	For the specified chain of <b>mBlk-clBlk</b> -cluster constructs, this routine frees all the <b>mBlk</b> structures back to the specified memory pool. It also decrements the reference count in all the <b>clBlk</b> structures. If the reference count in a <b>clBlk</b> falls to zero, this routine also frees that <b>clBlk</b> and its associated cluster back to the specified memory pool.
<b>VXWORKS AE PROTECTION DOMAINS</b>	Under VxWorks AE, you can call this function from within the kernel protection domain only. In addition, all arguments to this function can reference only that data which is valid in the kernel protection domain. This restriction does not apply under non-AE versions of VxWorks.
<b>RETURNS</b>	N/A
<b>ERRNO</b>	<code>S_netBufLib_MBLK_INVALID</code>
<b>SEE ALSO</b>	<code>netBufLib</code>

---

## netMblkClFree()

<b>NAME</b>	<code>netMblkClFree()</code> – free an <b>mBlk-clBlk</b> -cluster construct
<b>SYNOPSIS</b>	<pre>M_BLK_ID netMblkClFree (     M_BLK_ID pMblk          /* pointer to the mBlk */ )</pre>
<b>DESCRIPTION</b>	For the specified <b>mBlk-clBlk</b> -cluster construct, this routine frees the <b>mBlk</b> back to the specified memory pool. It also decrements the reference count in the <b>clBlk</b> structure. If the reference count falls to zero, no other <b>mBlk</b> structure reference this <b>clBlk</b> . In that case, this routine also frees the <b>clBlk</b> structure and its associated cluster back to the specified memory pool.

#### VXWORKS AE PROTECTION DOMAINS

Under VxWorks AE, you can call this function from within the kernel protection domain only. In addition, all arguments to this function can reference only that data which is valid in the kernel protection domain. Likewise, the returned ID is valid in the kernel protection domain only. This restriction does not apply under non-AE versions of VxWorks.

**RETURNS** If the specified **mBlk** was part of an **mBlk** chain, this routine returns a pointer to the next **mBlk**. Otherwise, it returns a **NULL**.

**ERRNO** **S\_netBufLib\_MBLK\_INVALID**

**SEE ALSO** **netBufLib**

---

## netMblkClGet()

**NAME** **netMblkClGet()** – get a **clBlk**-cluster and join it to the specified **mBlk**

**SYNOPSIS** **STATUS netMblkClGet**

```
(  
    NET_POOL_ID pNetPool,    /* pointer to the net pool */  
    M_BLK_ID    pMblk,      /* mBlk to embed the cluster in */  
    int         bufSize,    /* size of the buffer to get */  
    int         canWait,    /* wait or dontwait */  
    BOOL        bestFit     /* TRUE/FALSE */  
)
```

**DESCRIPTION** This routine gets a **clBlk**-cluster pair from the specified memory pool and joins it to the specified **mBlk** structure. The **mBlk-clBlk**-cluster triplet it produces is the basic structure for handling data at all layers of the network stack.

*pNetPool*

Expects a pointer to the memory pool from which you want to get a free **clBlk**-cluster pair.

*pMblk*

Expects a pointer to the **mBlk** structure (previously allocated) to which you want to join the retrieved **clBlk**-cluster pair.

*bufSize*

Expects the size, in bytes, of the cluster in the **clBlk**-cluster pair.

*canWait*

Expects either **M\_WAIT** or **M\_DONTWAIT**. If either item is not immediately available, the **M\_WAIT** value allows this routine to repeat the allocation attempt after performing garbage collection. It omits those steps when the **M\_DONTWAIT** value is used.

*bestFit*

Expects either **TRUE** or **FALSE**. If *bestFit* is **TRUE** and a cluster of the exact size is unavailable, this routine gets a larger cluster (if available). If *bestFit* is **FALSE** and an exact size cluster is unavailable, this routine gets either a smaller or a larger cluster (depending on what is available). Otherwise, it returns immediately with an **ERROR** value. For memory pools containing only one cluster size, *bestFit* should always be set to **FALSE**.

**VXWORKS AE PROTECTION DOMAINS**

Under VxWorks AE, you can call this function from within the kernel protection domain only. In addition, all arguments to this function can reference only that data which is valid in the kernel protection domain. This restriction does not apply under non-AE versions of VxWorks.

<b>RETURNS</b>	OK or ERROR.
<b>ERRNO</b>	S_netBufLib_CLSIZE_INVALID
<b>SEE ALSO</b>	netBufLib

---

## netMblkClJoin()

<b>NAME</b>	netMblkClJoin() – join an <b>mBlk</b> to a <b>clBlk</b> -cluster construct
<b>SYNOPSIS</b>	<pre> <b>M_BLK_ID</b> netMblkClJoin (     <b>M_BLK_ID</b> pMblk,           /* pointer to an mBlk */     <b>CL_BLK_ID</b> pClBlk         /* pointer to a cluster Blk */ ) </pre>
<b>DESCRIPTION</b>	This routine joins the previously reserved <b>mBlk</b> referenced in <i>pMblk</i> to the <b>clBlk</b> -cluster construct referenced in <i>pClBlk</i> . Internally, this routine sets the <b>M_EXT</b> flag in <b>mBlk.mBlkHdr.mFlags</b> . It also and sets the <b>mBlk.mBlkHdr.mData</b> to point to the start of the data in the cluster.

**VXWORKS AE PROTECTION DOMAINS**

Under VxWorks AE, you can call this function from within the kernel protection domain only. In addition, all arguments to this function can reference only that data which is valid in the kernel protection domain. Likewise, the returned ID is valid in the kernel protection domain only. This restriction does not apply under non-AE versions of VxWorks.

**RETURNS** M\_BLK\_ID or NULL.

**SEE ALSO** netBufLib

---

## netMblkDup()

**NAME** netMblkDup() – duplicate an mBlk

**SYNOPSIS**

```
M_BLK_ID netMblkDup
(
    M_BLK_ID pSrcMblk,          /* pointer to source mBlk */
    M_BLK_ID pDestMblk        /* pointer to the destination mBlk */
)
```

**DESCRIPTION** This routine copies the references from a source **mBlk** in an **mBlk-clBlk**-cluster construct to a stand-alone **mBlk**. This lets the two **mBlk** structures share the same **clBlk**-cluster construct. This routine also increments the reference count in the shared **clBlk**. The *pSrcMblk* expects a pointer to the source **mBlk**. The *pDestMblk* parameter expects a pointer to the destination **mBlk**.

**VXWORKS AE PROTECTION DOMAINS**

Under VxWorks AE, you can call this function from within the kernel protection domain only. In addition, all arguments to this function can reference only that data which is valid in the kernel protection domain. Likewise, the returned ID is valid in the kernel protection domain only. This restriction does not apply under non-AE versions of VxWorks.

**RETURNS** A pointer to the destination **mBlk** or NULL if the source **mBlk** referenced in *pSrcMblk* is not part of a valid **mBlk-clBlk**-cluster construct.

**SEE ALSO** netBufLib

---

## netMblkFree()

**NAME** netMblkFree() – free an **mBlk** back to its memory pool

**SYNOPSIS**

```
void netMblkFree
(
    NET_POOL_ID pNetPool,    /* pointer to the net pool */
    M_BLK_ID    pMblk        /* mBlk to free */
)
```

**DESCRIPTION** This routine frees the specified **mBlk** back to the specified memory pool.

### VXWORKS AE PROTECTION DOMAINS

Under VxWorks AE, you can call this function from within the kernel protection domain only. In addition, all arguments to this function can reference only that data which is valid in the kernel protection domain. This restriction does not apply under non-AE versions of VxWorks.

**RETURNS** N/A

**SEE ALSO** netBufLib

---

## netMblkGet()

**NAME** netMblkGet() – get an **mBlk** from a memory pool

**SYNOPSIS**

```
M_BLK_ID netMblkGet
(
    NET_POOL_ID pNetPool,    /* pointer to the net pool */
    int         canWait,     /* M_WAIT/M_DONTWAIT */
    UCHAR       type        /* mBlk type */
)
```

**DESCRIPTION** This routine allocates an **mBlk** from the specified memory pool, if available.

*pNetPool*

Expects a pointer to the pool from which you want an **mBlk**.

*canWait*

Expects either **M\_WAIT** or **M\_DONTWAIT**. If no **mBlk** is immediately available, the

**M\_WAIT** value allows this routine to repeat the allocation attempt after performing garbage collection. It omits these steps when the **M\_DONTWAIT** value is used.

*type*

Expects the type value that you want to associate with the returned **mBlk**.

#### **VXWORKS AE PROTECTION DOMAINS**

Under VxWorks AE, you can call this function from within the kernel protection domain only. In addition, all arguments to this function can reference only that data which is valid in the kernel protection domain. Likewise, the returned ID is valid in the kernel protection domain only. This restriction does not apply under non-AE versions of VxWorks.

**RETURNS** **M\_BLK\_ID** or **NULL** if no **mBlk** is available.

**ERRNO** **S\_netBufLib\_MBLK\_INVALID**

**SEE ALSO** **netBufLib**

---

## **netMblkToBufCopy()**

**NAME** **netMblkToBufCopy()** – copy data from an **mBlk** to a buffer

**SYNOPSIS**

```
int netMblkToBufCopy
(
    M_BLK_ID pMblk,           /* pointer to an mBlk */
    char *   pBuf,           /* pointer to the buffer to copy */
    FUNCPTR  pCopyRtn        /* function pointer for copy routine */
)
```

**DESCRIPTION** This routine copies data from the **mBlk** chain referenced in *pMblk* to the buffer referenced in *pBuf*. It is assumed that *pBuf* points to enough memory to contain all the data in the entire **mBlk** chain. The argument *pCopyRtn* expects either a **NULL** or a function pointer to a copy routine. The arguments passed to the copy routine are source pointer, destination pointer and the length of data to copy. If *pCopyRtn* is **NULL**, **netMblkToBufCopy()** uses a default routine to extract the data from the chain.

**RETURNS** The length of data copied or zero.

**SEE ALSO** **netBufLib**

---

## netPoolDelete()

**NAME** `netPoolDelete()` – delete a memory pool

**SYNOPSIS**

```
STATUS netPoolDelete
(
    NET_POOL_ID pNetPool    /* pointer to a net pool */
)
```

**DESCRIPTION** This routine deletes the specified `netBufLib`-managed memory pool.

### VXWORKS AE PROTECTION DOMAINS

Under VxWorks AE, you can call this function from within the kernel protection domain only. In addition, all arguments to this function can reference only that data which is valid in the kernel protection domain. This restriction does not apply under non-AE versions of VxWorks.

**RETURNS** OK or ERROR.

**ERRNO** `S_netBufLib_NETPOOL_INVALID`

**SEE ALSO** `netBufLib`

---

## netPoolInit()

**NAME** `netPoolInit()` – initialize a `netBufLib`-managed memory pool

**SYNOPSIS**

```
STATUS netPoolInit
(
    NET_POOL_ID pNetPool,    /* pointer to a net pool */
    M_CL_CONFIG * pMclBlkConfig, /* pointer to a mBlk configuration */
    CL_DESC * pClDescTbl,    /* pointer to cluster desc table */
    int c1DescTblNumEnt,    /* number of cluster desc entries */
    POOL_FUNC * pFuncTbl    /* pointer to pool function table */
)
```

**DESCRIPTION** Call this routine to set up a `netBufLib`-managed memory pool. Within this pool, `netPoolInit()` organizes several sub-pools: one for `mBlk` structures, one for `clBlk` structures, and as many cluster sub-pools as there are cluster sizes. As input, this routine expects the following parameters:

*pNetPool*

Expects a **NET\_POOL\_ID** that points to a previously allocated **NET\_POOL** structure. You need not initialize any values in this structure. That is handled by **netPoolInit()**.

*mClBlkConfig*

Expects a pointer to a previously allocated and initialized **M\_CL\_CONFIG** structure. Within this structure, you must provide four values: **mBlkNum**, a count of **mBlk** structures; **cBlkNum**, a count of **cBlk** structures; **memArea**, a pointer to an area of memory that can contain all the **mBlk** and **cBlk** structures; and **memSize**, the size of that memory area. For example, you can set up an **M\_CL\_CONFIG** structure as follows:

```
M_CL_CONFIG mClBlkConfig = /* mBlk, cBlk configuration table */
{
    mBlkNum      cBlkNum      memArea      memSize
    -----    ----        -
    400,         245,         0xfe000000,  21260
};
```

You can calculate the **memArea** and **memSize** values. Such code could first define a table as shown above, but set both **memArea** and **memSize** as follows:

```
mClBlkConfig.memSize = (mClBlkConfig.mBlkNum * (M_BLK_SZ + sizeof(long))) +
                        (mClBlkConfig.cBlkNum * CL_BLK_SZ);
```

You can set the **memArea** value to a pointer to private memory, or you can reserve the memory with a call to **malloc()**. For example:

```
mClBlkConfig.memArea = malloc(mClBlkConfig.memSize);
```

The **netBufLib.h** file defines **M\_BLK\_SZ** as:

```
sizeof(struct mBlk)
```

Currently, this evaluates to 32 bytes. Likewise, this file defines **CL\_BLK\_SZ** as:

```
sizeof(struct cBlk)
```

Currently, this evaluates to 32 bytes.

When choosing values for **mBlkNum** and **cBlkNum**, remember that you need as many **cBlk** structures as you have clusters (data buffers). You also need at least as many **mBlk** structures as you have **cBlk** structures, but you will most likely need more. That is because **netBufLib** shares buffers by letting multiple **mBlk** structures join to the same **cBlk** and thus to its underlying cluster. The **cBlk** keeps a count of the number of **mBlk** structures that reference it.

*pClDescTbl*

Expects a pointer to a table of previously allocated and initialized **CL\_DESC** structures. Each structure in this table describes a single cluster pool. You need a dedicated cluster pool for each cluster size you want to support. Within each **CL\_DESC** structure, you must provide four values: **clusterSize**, the size of a cluster in

this cluster pool; **num**, the number of clusters in this cluster pool; **memArea**, a pointer to an area of memory that can contain all the clusters; and **memSize**, the size of that memory area.

Thus, if you need to support six different cluster sizes, this parameter must point to a table containing six **CL\_DESC** structures. For example, consider the following:

```
CL_DESC clDescTbl [] = /* cluster descriptor table */
{
/*
clusterSize      num      memArea      memSize
-----
*/
{64,             100,     0x10000,     6800},
{128,            50,      0x20000,     6600},
{256,            50,      0x30000,    13000},
{512,            25,      0x40000,    12900},
{1024,           10,      0x50000,    10280},
{2048,           10,      0x60000,    20520}
};
```

As with the **memArea** and **memSize** members in the **M\_CL\_CONFIG** structure, you can set these members of the **CL\_DESC** structures by calculation after you create the table. The formula would be as follows:

```
clDescTbl[n].memSize =
    (clDescTbl[n].num * (clDescTbl[n].clusterSize + sizeof(long)));
```

The **memArea** member can point to a private memory area that you know to be available for storing clusters, or you can use **malloc()**.

```
clDescTbl[n].memArea = malloc( clDescTbl[n].memSize );
```

Valid cluster sizes range from 64 bytes to 65536 bytes. If there are multiple cluster pools, valid sizes are further restricted to powers of two (for example, 64, 128, 256, and so on). If there is only one cluster pool (as is often the case for the memory pool specific to a single device driver), there is no power of two restriction. Thus, the cluster can be of any size between 64 bytes and 65536 bytes on 4-byte alignment. A typical buffer size for Ethernet devices is 1514 bytes. However, because a cluster size requires a 4-byte alignment, the cluster size for this Ethernet buffer would have to be increased to at least 1516 bytes.

*clDescTblNumEnt*

Expects a count of the elements in the **CL\_DESC** table referenced by the *pClDescTbl* parameter. This is a count of the number of cluster pools. You can get this value using the **NELEMENTS** macro defined in **vxWorks.h**. For example:

```
int clDescTblNumEnt = (NELEMENTS(clDescTbl));
```

*pFuncTbl*

Expects a **NULL** or a pointer to a function table. This table contains pointers to the

functions used to manage the buffers in this memory pool. Using a NULL for this parameter tells **netBufLib** to use its default function table. If you opt for the default function table, every **mBlk** and every cluster is prepended by a 4-byte header (which is why the size calculations above for clusters and **mBlk** structures contained an extra **sizeof(long)**). However, users need not concern themselves with this header when accessing these buffers. The returned pointers from functions such as **netClusterGet()** return pointers to the start of data, which is just after the header.

Assuming you have set up the configuration tables as shown above, a typical call to **netPoolInit()** would be as follows:

```
int c1DescTblNumEnt = (NELEMENTS(c1DescTbl));
NET_POOL netPool;
NET_POOL_ID pNetPool = &netPool;
if (netPoolInit (pNetPool, &mClBlkConfig, &c1DescTbl [0],
c1DescTblNumEnt,
NULL) != OK)
return (ERROR);
```

#### VXWORKS AE PROTECTION DOMAINS

Under VxWorks AE, access to the contents of a memory pool is limited to the protection domain within which you made the **netPoolInit()** call that created the pool. In addition, all parameters to a **netPoolInit()** call must be valid within the protection domain from which you make the call. This restriction does not apply under non-AE versions of VxWorks.

**RETURNS** OK or ERROR.

**ERRNO** S\_netBufLib\_MEMSIZE\_INVALID  
S\_netBufLib\_CLSIZE\_INVALID  
S\_netBufLib\_NO\_SYSTEM\_MEMORY  
S\_netBufLib\_MEM\_UNALIGNED  
S\_netBufLib\_MEMSIZE\_UNALIGNED  
S\_netBufLib\_MEMAREA\_INVALID

**SEE ALSO** netBufLib, netPoolDelete()

---

## netPoolKheapInit()

<b>NAME</b>	<code>netPoolKheapInit()</code> – kernel heap version of <code>netPoolInit()</code>
<b>SYNOPSIS</b>	<pre>STATUS netPoolKheapInit (     NET_POOL_ID  pNetPool,          /* pointer to a net pool */     M_CL_CONFIG * pMclBlkConfig,   /* pointer to a mBlk configuration */     CL_DESC *    pClDescTbl,       /* pointer to cluster desc table */     int          clDescTblNumEnt,  /* number of cluster desc entries */     POOL_FUNC *  pFuncTbl          /* pointer to pool function table */ )</pre>
<b>DESCRIPTION</b>	This initializes a <code>netBufLib</code> -managed memory pool from Kernel heap. See <code>netPoolInit()</code> for more detail.
<b>VXWORKS AE PROTECTION DOMAINS</b>	Under VxWorks AE, you can call this function from within the kernel protection domain only. In addition, all arguments to this function can reference only that data which is valid in the kernel protection domain. This restriction does not apply under non-AE versions of VxWorks.
<b>RETURNS</b>	OK or ERROR.
<b>ERRNO</b>	N/A
<b>SEE ALSO</b>	<code>netBufLib</code> , <code>netPoolInit()</code> , <code>netPoolDelete()</code>

---

## netPoolShow()

<b>NAME</b>	<code>netPoolShow()</code> – show pool statistics
<b>SYNOPSIS</b>	<pre>void netPoolShow (     NET_POOL_ID pNetPool )</pre>
<b>DESCRIPTION</b>	This routine displays the distribution of <code>mBlks</code> and clusters in a given network pool ID.

**EXAMPLE**

```
void endPoolShow
(
    char * devName,      /* The interface name: "dc", "ln" ...*/
    int   unit          /* the unit number: usually 0      */
)
{
    END_OBJ * pEnd;
    if ((pEnd = endFindByName (devName, unit)) != NULL)
        netPoolShow (pEnd->pNetPool);
    else
        printf ("Could not find device %s\n", devName);
    return;
}
```

**RETURNS** N/A

**SEE ALSO** netShow

---

## netShowInit()

**NAME** netShowInit() – initialize network show routines

**SYNOPSIS** void netShowInit (void)

**DESCRIPTION** This routine links the network show facility into the VxWorks system. These routines are included automatically if `INCLUDE_NET_SHOW` is defined.

**RETURNS** N/A

**SEE ALSO** netShow

---

## netStackDataPoolShow()

<b>NAME</b>	<code>netStackDataPoolShow()</code> – show network stack data pool statistics
<b>SYNOPSIS</b>	<code>void netStackDataPoolShow (void)</code>
<b>DESCRIPTION</b>	<p>This routine displays the distribution of <b>mBlks</b> and clusters in a the network data pool. The network data pool is used only for data transfer through the network stack.</p> <p>The “clusters” column indicates the total number of clusters of that size that have been allocated. The “free” column indicates the number of available clusters of that size (the total number of clusters minus those clusters that are in use). The “usage” column indicates the number of times clusters have been allocated (not, as you might expect, the number of clusters currently in use).</p>
<b>RETURNS</b>	N/A
<b>SEE ALSO</b>	<code>netShow</code> , <code>netStackSysPoolShow()</code> , <code>netBufLib</code>

---

## netStackSysPoolShow()

<b>NAME</b>	<code>netStackSysPoolShow()</code> – show network stack system pool statistics
<b>SYNOPSIS</b>	<code>void netStackSysPoolShow (void)</code>
<b>DESCRIPTION</b>	<p>This routine displays the distribution of <b>mBlks</b> and clusters in a the network system pool. The network system pool is used only for system structures such as sockets, routes, interface addresses, protocol control blocks, multicast addresses, and multicast route entries.</p> <p>The “clusters” column indicates the total number of clusters of that size that have been allocated. The “free” column indicates the number of available clusters of that size (the total number of clusters minus those clusters that are in use). The “usage” column indicates the number of times clusters have been allocated (not, as you might expect, the number of clusters currently in use).</p>
<b>RETURNS</b>	N/A
<b>SEE ALSO</b>	<code>netShow</code> , <code>netStackDataPoolShow()</code> , <code>netBufLib</code>

---

## netTask()

**NAME** `netTask()` – network task entry point

**SYNOPSIS** `void netTask (void)`

**DESCRIPTION** This routine is the VxWorks network support task. Most of the VxWorks network runs in this task's context.

---

**NOTE:** To prevent an application task from monopolizing the CPU if it is in an infinite loop or is never blocked, the priority of `netTask()` relative to an application may need to be adjusted. Network communication may be lost if `netTask()` is “starved” of CPU time. The default task priority of `netTask()` is 50. Use `taskPrioritySet()` to change the priority of a task.

---

This task is spawned by `netLibInit()`.

### VXWORKS AE PROTECTION DOMAINS

Under VxWorks AE, you can call this function from within the kernel protection domain only. This restriction does not apply under non-AE versions of VxWorks.

**RETURNS** N/A

**SEE ALSO** `netLib`, `netLibInit()`

---

## netTupleGet()

**NAME** `netTupleGet()` – get an `mBlk-clBlk`-cluster

**SYNOPSIS**

```
M_BLK_ID netTupleGet
(
    NET_POOL_ID pNetPool,    /* pointer to the net pool */
    int         bufSize,     /* size of the buffer to get */
    int         canWait,     /* wait or dontwait */
    UCHAR       type,        /* type of data */
    BOOL        bestFit      /* TRUE/FALSE */
)
```

**DESCRIPTION** This routine gets an `mBlk-clBlk`-cluster triplet from the specified memory pool. The resulting structure is the basic method for accessing data at all layers of the network stack.

*pNetPool*

Expects a pointer to the memory pool with which you want to build a **mBlk-clBlk**-cluster triplet.

*bufSize*

Expects the size, in bytes, of the cluster in the **clBlk**-cluster pair.

*canWait*

Expects either **M\_WAIT** or **M\_DONTWAIT**. If any item in the triplet is not immediately available, the **M\_WAIT** value allows this routine to repeat the allocation attempt after performing garbage collection. The **M\_DONTWAIT** value prevents those extra steps.

*type*

Expects the type of data, for example **MT\_DATA**, **MT\_HEADER**. The various values for this type are defined in **netBufLib.h**.

*bestFit*

Expects either **TRUE** or **FALSE**. If *bestFit* is **TRUE** and a cluster of the exact size is unavailable, this routine gets a larger cluster (if available). If *bestFit* is **FALSE** and an exact size cluster is unavailable, this routine gets either a smaller or a larger cluster (depending on what is available). Otherwise, it returns immediately with an **ERROR** value. For memory pools containing only one cluster size, *bestFit* should always be set to **FALSE**.

#### **VXWORKS AE PROTECTION DOMAINS**

Under VxWorks AE, you can call this function from within the kernel protection domain only. In addition, all arguments to this function can reference only that data which is valid in the kernel protection domain. Likewise, the returned ID is valid in the kernel protection domain only. This restriction does not apply under non-AE versions of VxWorks.

<b>RETURNS</b>	<b>M_BLK_ID</b> or <b>NULL</b> .
<b>ERRNO</b>	<b>S_netBufLib_MBLK_INVALID</b> <b>S_netBufLib_CLSIZE_INVALID</b> <b>S_netBufLib_NETPOOL_INVALID</b>
<b>SEE ALSO</b>	<b>netBufLib</b>

---

## nextIndex()

<b>NAME</b>	<b>nextIndex()</b> – the comparison routine for the AVL tree
<b>SYNOPSIS</b>	<pre>int nextIndex (     void *          pAvlNode, /* The node to compare with */     GENERIC_ARGUMENT key      /* The given index */ )</pre>
<b>DESCRIPTION</b>	This routine compares the two indexes and returns a code based on whether the index, in question, is lesser than, equal to or greater than the one being compared.
<b>RETURNS</b>	-1, if the given index is lesser; 0, if equal; and 1, if greater.
<b>SEE ALSO</b>	<b>m2IfLib</b>

---

## nfsAuthUnixGet()

<b>NAME</b>	<b>nfsAuthUnixGet()</b> – get the NFS UNIX authentication parameters
<b>SYNOPSIS</b>	<pre>void nfsAuthUnixGet (     char * machname,          /* where to store host machine */     int *  pUid,              /* where to store user ID */     int *  pGid,              /* where to store group ID */     int *  pNgids,           /* where to store number of group IDs */     int *  gids               /* where to store array of group IDs */ )</pre>
<b>DESCRIPTION</b>	This routine gets the previously set UNIX authentication values.
<b>RETURNS</b>	N/A
<b>SEE ALSO</b>	<b>nfsLib</b> , <b>nfsAuthUnixPrompt()</b> , <b>nfsAuthUnixShow()</b> , <b>nfsAuthUnixSet()</b> , <b>nfsIdSet()</b>

---

## nfsAuthUnixPrompt()

**NAME** `nfsAuthUnixPrompt()` – modify the NFS UNIX authentication parameters

**SYNOPSIS** `void nfsAuthUnixPrompt (void)`

**DESCRIPTION** This routine allows UNIX authentication parameters to be changed from the shell. The user is prompted for each parameter, which can be changed by entering the new value next to the current one.

**EXAMPLE**

```
-> nfsAuthUnixPrompt
machine name:  yuba
user ID:      2001 128
group ID:     100
num of groups: 1 3
group #1:    100 100
group #2:     0 120
group #3:     0 200
value = 3 = 0x3
```

**SEE ALSO** `nfsLib`, `nfsAuthUnixShow()`, `nfsAuthUnixSet()`, `nfsAuthUnixGet()`, `nfsIdSet()`

N

---

## nfsAuthUnixSet()

**NAME** `nfsAuthUnixSet()` – set the NFS UNIX authentication parameters

**SYNOPSIS**

```
void nfsAuthUnixSet
(
    char * machname,          /* host machine */
    int   uid,                /* user ID */
    int   gid,                /* group ID */
    int   ngids,              /* number of group IDs */
    int * aup_gids            /* array of group IDs */
)
```

**DESCRIPTION** This routine sets UNIX authentication parameters. It is initially called by `usrNetInit()`. *machname* should be set with the name of the mounted system (i.e., the target name itself) to distinguish hosts from hosts on a NFS network.

**RETURNS** N/A

**SEE ALSO** `nfsLib`, `nfsAuthUnixPrompt()`, `nfsAuthUnixShow()`, `nfsAuthUnixGet()`, `nfsIdSet()`

---

## nfsAuthUnixShow()

**NAME** `nfsAuthUnixShow()` – display the NFS UNIX authentication parameters

**SYNOPSIS** `void nfsAuthUnixShow (void)`

**DESCRIPTION** This routine displays the parameters set by `nfsAuthUnixSet()` or `nfsAuthUnixPrompt()`.

**EXAMPLE**

```
-> nfsAuthUnixShow
machine name = yuba
user ID      = 2001
group ID     = 100
group [0]    = 100
value = 1 = 0x1
```

**RETURNS** N/A

**SEE ALSO** `nfsLib`, `nfsAuthUnixPrompt()`, `nfsAuthUnixSet()`, `nfsAuthUnixGet()`, `nfsIdSet()`

---

## nfsDevInfoGet()

**NAME** `nfsDevInfoGet()` – read configuration information from the requested NFS device

**SYNOPSIS** `STATUS nfsDevInfoGet`

```
(
    unsigned long  nfsDevHandle, /* NFS device handle */
    NFS_DEV_INFO * pnfsInfo      /* ptr to struct to hold config info */
)
```

**DESCRIPTION** This routine accesses the NFS device specified in the parameter `nfsDevHandle` and fills in the structure pointed to by `pnfsInfo`. The calling function should allocate memory for `pnfsInfo` and for the two character buffers, `remFileSys` and `locFileSys`, that are part of `pnfsInfo`. These buffers should have a size of `nfsMaxPath`.

**RETURNS** OK if *nfsInfo* information is valid, otherwise **ERROR**.

**SEE ALSO** `nfsDrv`, `nfsDevListGet()`

---

## nfsDevListGet()

**NAME** `nfsDevListGet()` – create list of all the NFS devices in the system

**SYNOPSIS**

```
int nfsDevListGet
(
    unsigned long nfsDevList[], /* NFS dev list of handles */
    int          listSize      /* number of elements available in list */
)
```

**DESCRIPTION** This routine fills the array *nfsDevlist* up to *listSize*, with handles to NFS devices currently in the system.

**RETURNS** The number of entries filled in the *nfsDevList* array.

**SEE ALSO** `nfsDrv`, `nfsDevInfoGet()`

---

## nfsDevShow()

**NAME** `nfsDevShow()` – display the mounted NFS devices

**SYNOPSIS** `void nfsDevShow (void)`

**DESCRIPTION** This routine displays the device names and their associated NFS file systems.

**EXAMPLE**

```
-> nfsDevShow
device name      file system
-----
/yuba1/          yuba:/yuba1
/wrs1/           wrs:/wrs1
```

**RETURNS** N/A

**SEE ALSO** `nfsDrv`

---

## nfsdInit()

**NAME** `nfsdInit()` – initialize the NFS server

**SYNOPSIS**

```
STATUS nfsdInit
(
    int      nServers,          /* the number of NFS servers to create */
    int      nExportedFs,      /* maximum number of exported file systems */
    int      priority,         /* the priority for the NFS servers */
    FUNCPTR  authHook,         /* authentication hook */
    FUNCPTR  mountAuthHook,    /* authentication hook for mount daemon */
    int      options           /* currently unused */
)
```

**DESCRIPTION** This routine initializes the NFS server. *nServers* specifies the number of tasks to be spawned to handle NFS requests. *priority* is the priority that those tasks will run at. *authHook* is a pointer to an authorization routine. *mountAuthHook* is a pointer to a similar routine, passed to `mountdInit()`. *options* is provided for future expansion.

Normally, no authorization is performed by either `mountd` or `nfsd`. If you want to add authorization, set *authHook* to a function pointer to a routine declared as follows:

```
nfsstat routine
(
    int          progNum,      /* RPC program number */
    int          versNum,     /* RPC program version number */
    int          procNum,     /* RPC procedure number */
    struct sockaddr_in clientAddr, /* address of the client */
    NFSD_ARGUMENT *  nfsdArg  /* argument of the call */
)
```

The *authHook* routine should return `NFS_OK` if the request is authorized, and `NFSERR_ACCES` if not. (`NFSERR_ACCES` is not required; any legitimate NFS error code can be returned.)

See `mountdInit()` for documentation on *mountAuthHook*. Note that *mountAuthHook* and *authHook* can point to the same routine. Simply use the *progNum*, *versNum*, and *procNum* fields to decide whether the request is an NFS request or a `mountd` request.

### VXWORKS AE PROTECTION DOMAINS

Under VxWorks AE, you can call this function from within the kernel protection domain only. In addition, all arguments to this function can reference only that data which is valid in the kernel protection domain. This restriction does not apply under non-AE versions of VxWorks.

**RETURNS** OK, or **ERROR** if the NFS server cannot be started.

**SEE ALSO** `nfsdLib`, `nfsExport()`, `mountdInit()`

---

## **nfsDrv()**

**NAME** `nfsDrv()` – install the NFS driver

**SYNOPSIS** `STATUS nfsDrv (void)`

**DESCRIPTION** This routine initializes and installs the NFS driver. It must be called before any reads, writes, or other NFS calls. This is done automatically when `INCLUDE_NFS` is defined.

### **VXWORKS AE PROTECTION DOMAINS**

Under VxWorks AE, you can call this function from within the kernel protection domain only. This restriction does not apply under non-AE versions of VxWorks.

**RETURNS** OK, or **ERROR** if there is no room for the driver.

**SEE ALSO** `nfsDrv`

---

## **nfsDrvNumGet()**

**NAME** `nfsDrvNumGet()` – return the IO system driver number for the NFS driver

**SYNOPSIS** `int nfsDrvNumGet (void)`

**DESCRIPTION** This routine returns the NFS driver number allocated by `iosDrvInstall()` during the NFS driver initialization. If the NFS driver has yet to be initialized, or if initialization failed, `nfsDrvNumGet()` will return **ERROR**.

**RETURNS** the NFS driver number or **ERROR**

**SEE ALSO** `nfsDrv`

## **nfsdStatusGet()**

**NAME** `nfsdStatusGet()` – get the status of the NFS server

**SYNOPSIS**

```
STATUS nfsdStatusGet  
(  
    NFS_SERVER_STATUS * serverStats /* pointer to status structure */  
)
```

**DESCRIPTION** This routine gets status information about the NFS server.

**RETURNS** OK, or ERROR if the information cannot be obtained.

**SEE ALSO** `nfsdLib`

---

## **nfsdStatusShow()**

**NAME** `nfsdStatusShow()` – show the status of the NFS server

**SYNOPSIS**

```
STATUS nfsdStatusShow  
(  
    int options /* unused */  
)
```

**DESCRIPTION** This routine shows status information about the NFS server.

**RETURNS** OK, or ERROR if the information cannot be obtained.

**SEE ALSO** `nfsdLib`

---

## nfsExport()

**NAME** `nfsExport()` – specify a file system to be NFS exported

**SYNOPSIS**

```
STATUS nfsExport
(
    char * directory,          /* Directory to export - FS must support NFS */
    int   id,                 /* ID number for file system */
    BOOL  readOnly,          /* TRUE if file system is exported read-only */
    int   options              /* Reserved for future use - set to 0 */
)
```

**DESCRIPTION** This routine makes a file system available for mounting by a client. The client should be in the local host table (see `hostAdd()`), although this is not required.

The *id* parameter can either be set to a specific value, or to 0. If it is set to 0, an ID number is assigned sequentially. Every time a file system is exported, it must have the same ID number, or clients currently mounting the file system will not be able to access files.

To display a list of exported file systems, use:

```
-> nfsExportShow "localhost"
```

**RETURNS** OK, or **ERROR** if the file system could not be exported.

**SEE ALSO** `mountLib`, `nfsLib`, `nfsExportShow()`, `nfsUnexport()`

---

## nfsExportShow()

**NAME** `nfsExportShow()` – display the exported file systems of a remote host

**SYNOPSIS**

```
STATUS nfsExportShow
(
    char * hostName           /* host machine to show exports for */
)
```

**DESCRIPTION** This routine displays the file systems of a specified host and the groups that are allowed to mount them.

**EXAMPLE**

```
-> nfsExportShow "wrs"
/d0                staff
```

**nfsHelp()**

```

/d1          staff eng
/d2          eng
/d3
value = 0 = 0x0

```

**RETURNS** OK or ERROR.

**SEE ALSO** nfsLib

---

## nfsHelp()

**NAME** nfsHelp() – display the NFS help menu

**SYNOPSIS** void nfsHelp (void)

**DESCRIPTION** This routine displays a summary of NFS facilities typically called from the shell:

```

nfsHelp          Print this list
netHelp          Print general network help list
nfsMount "host","filesystem",["devname"] Create device with
                file system/directory from host
nfsUnmount "devname" Remove an NFS device
nfsAuthUnixShow  Print current UNIX authentication
nfsAuthUnixPrompt Prompt for UNIX authentication
nfsIdSet id      Set user ID for UNIX authentication
nfsDevShow       Print list of NFS devices
nfsExportShow "host" Print a list of NFS file systems which
                are exported on the specified host
mkdir "dirname"  Create directory
rm "file"        Remove file
EXAMPLE: -> hostAdd "wrs", "90.0.0.2"
        -> nfsMount "wrs", "/disk0/path/mydir", "/mydir/"
        -> cd "/mydir/"
        -> nfsAuthUnixPrompt /* fill in user ID, etc. */
        -> ls /* list /disk0/path/mydir */
        -> copy < foo /* copy foo to standard out */
        -> ld < foo.o /* load object module foo.o */
        -> nfsUnmount "/mydir/" /* remove NFS device /mydir/ */

```

**RETURNS** N/A

**SEE ALSO** nfsLib

---

## nfsIdSet()

<b>NAME</b>	<b>nfsIdSet()</b> – set the ID number of the NFS UNIX authentication parameters
<b>SYNOPSIS</b>	<pre>void nfsIdSet (     int uid                /* user ID on host machine */ )</pre>
<b>DESCRIPTION</b>	This routine sets only the UNIX authentication user ID number. For most NFS permission needs, only the user ID needs to be changed. Set <i>uid</i> to the user ID on the NFS server.
<b>RETURNS</b>	N/A
<b>SEE ALSO</b>	<b>nfsLib</b> , <b>nfsAuthUnixPrompt()</b> , <b>nfsAuthUnixShow()</b> , <b>nfsAuthUnixSet()</b> , <b>nfsAuthUnixGet()</b>

---

## nfsMount()

<b>NAME</b>	<b>nfsMount()</b> – mount an NFS file system
<b>SYNOPSIS</b>	<pre>STATUS nfsMount (     char * host,            /* name of remote host */     char * fileSystem,     /* name of remote directory to mount */     char * localName       /* local device name for remote dir (NULL = */                           /* use fileSystem name) */ )</pre>
<b>DESCRIPTION</b>	This routine mounts a remote file system. It creates a local device <i>localName</i> for a remote file system on a specified host. The host must have already been added to the local host table with <b>hostAdd()</b> . If <i>localName</i> is <b>NULL</b> , the local name will be the same as the remote name.
<b>RETURNS</b>	<b>OK</b> , or <b>ERROR</b> if the driver is not installed, <i>host</i> is invalid, or memory is insufficient.
<b>SEE ALSO</b>	<b>nfsDrv</b> , <b>nfsUnmount()</b> , <b>hostAdd()</b>

## nfsMountAll()

<b>NAME</b>	<b>nfsMountAll()</b> – mount all file systems exported by a specified host
<b>SYNOPSIS</b>	<pre><b>STATUS</b> nfsMountAll (     char * pHostName, /* name of remote host */     char * pClientName, /* name of a client specified in access */                         /* list, if any */     <b>BOOL</b> quietFlag /* FALSE = print name of each mounted file system */ )</pre>
<b>DESCRIPTION</b>	<p>This routine mounts the file systems exported by the host <i>pHostName</i> which are accessible by <i>pClientName</i>. A <i>pClientName</i> entry of <b>NULL</b> will only mount file systems that are accessible by any client. The <b>nfsMount()</b> routine is called to mount each file system. It creates a local device for each mount that has the same name as the remote file system.</p> <p>If the <i>quietFlag</i> setting is <b>FALSE</b>, each file system is printed on standard output after it is mounted successfully.</p>
<b>RETURNS</b>	<b>OK</b> , or <b>ERROR</b> if any mount fails.
<b>SEE ALSO</b>	<b>nfsDrv</b> , <b>nfsMount()</b>

---

## nfsUnexport()

<b>NAME</b>	<b>nfsUnexport()</b> – remove a file system from the list of exported file systems
<b>SYNOPSIS</b>	<pre><b>STATUS</b> nfsUnexport (     char * dirName /* Name of the directory to unexport */ )</pre>
<b>DESCRIPTION</b>	<p>This routine removes a file system from the list of file systems exported from the target. Any client attempting to mount a file system that is not exported will receive an error (<b>NFSERR_ACCESS</b>).</p>
<b>RETURNS</b>	<b>OK</b> , or <b>ERROR</b> if the file system could not be removed from the exports list.
<b>ERRNO</b>	<b>ENOENT</b>
<b>SEE ALSO</b>	<b>mountLib</b> , <b>nfsLib</b> , <b>nfsExportShow()</b> , <b>nfsExport()</b>

---

## nfsUnmount()

- NAME** `nfsUnmount()` – unmount an NFS device
- SYNOPSIS**
- ```
STATUS nfsUnmount
(
    char * localName          /* local of nfs device */
)
```
- DESCRIPTION** This routine unmounts file systems that were previously mounted via NFS.
- RETURNS** OK, or ERROR if *localName* is not an NFS device or cannot be mounted.
- SEE ALSO** `nfsDrv`, `nfsMount()`

---

## ntPassFsDevInit()

- NAME** `ntPassFsDevInit()` – associate a device with ntPassFs file system functions
- SYNOPSIS**
- ```
void *ntPassFsDevInit
(
    char * devName           /* device name */
)
```
- DESCRIPTION** This routine associates the name *devName* with the file system and installs it in the I/O System's device table. The driver number used when the device is added to the table is that which was assigned to the ntPassFs library during `ntPassFsInit()`.
- RETURNS** A pointer to the volume descriptor, or NULL if there is an error.
- SEE ALSO** `ntPassFsLib`

## ntPassFsInit()

**NAME**            **ntPassFsInit()** – prepare to use the ntPassFs library

**SYNOPSIS**        **STATUS** **ntPassFsInit**  
                  (  
                  **int** **nPassfs**                    **/\* number of ntPass-through file systems \*/**  
                  )

**DESCRIPTION**    This routine initializes the ntPassFs library. It must be called exactly once, before any other routines in the library. The argument specifies the number of ntPassFs devices that may be open at once. This routine installs **ntPassFsLib** as a driver in the I/O system driver table, allocates and sets up the necessary memory structures, and initializes semaphores.

Normally this routine is called from the root task, **usrRoot()**, in **usrConfig()**. To enable this initialization, define **INCLUDE\_PASSFS** in **configAll.h**.

---

**NOTE:** Maximum number of ntPass-through file systems is 1.

---

**RETURNS**        **OK**, or **ERROR**.

**SEE ALSO**        **ntPassFsLib**

---

## open()

**NAME** open() – open a file

**SYNOPSIS**

```
int open
(
    const char * name,          /* name of the file to open */
    int         flags,         /* O_RDONLY, O_WRONLY, O_RDWR, or O_CREAT */
    int         mode           /* mode of file to create (UNIX chmod style) */
)
```

**DESCRIPTION** This routine opens a file for reading, writing, or updating, and returns a file descriptor for that file. The arguments to **open()** are the filename and the type of access:

**O\_RDONLY** (0) (or **READ**) - open for reading only.  
**O\_WRONLY** (1) (or **WRITE**) - open for writing only.  
**O\_RDWR** (2) (or **UPDATE**) - open for reading and writing.  
**O\_CREAT** (0x0200) - create a file.

In general, **open()** can only open pre-existing devices and files. However, for NFS network devices only, files can also be created with **open()** by performing a logical OR operation with **O\_CREAT** and the *flags* argument. In this case, the file is created with a UNIX **chmod**-style file mode, as indicated with *mode*. For example:

```
fd = open ("/usr/myFile", O_CREAT | O_RDWR, 0644);
```

Only the NFS driver uses the *mode* argument.

---

**NOTE:** For more information about situations when there are no file descriptors available, see the manual entry for **iosInit()**.

---

**RETURNS** A file descriptor number, or **ERROR** if a file name is not specified, the device does not exist, no file descriptors are available, or the driver returns **ERROR**.

**ERRNO** ELOOP

**SEE ALSO** ioLib, creat()

## opendir( )

**NAME**            **opendir( )** – open a directory for searching (POSIX)

**SYNOPSIS**        **DIR \*opendir**  
                  (  
                  **char \* dirName**                */\* name of directory to open \*/*  
                  )

**DESCRIPTION**    This routine opens the directory named by *dirName* and allocates a directory descriptor (DIR) for it. A pointer to the DIR structure is returned. The return of a NULL pointer indicates an error.

After the directory is opened, **readdir( )** is used to extract individual directory entries. Finally, **closedir( )** is used to close the directory.

---

**WARNING:** For remote file systems mounted over **netDrv**, **opendir( )** fails, because the **netDrv** implementation strategy does not provide a way to distinguish directories from plain files. To permit use of **opendir( )** on remote files, use NFS rather than **netDrv**.

---

**RETURNS**        A pointer to a directory descriptor, or NULL if there is an error.

**SEE ALSO**        **dirLib**, **closedir( )**, **readdir( )**, **rewinddir( )**, **ls( )**

---

## operator delete( )

**NAME**            **operator delete( )** – default run-time support for memory deallocation (C++)

**SYNOPSIS**        **extern void operator delete**  
                  (  
                  **void \* pMem**                */\* pointer to dynamically-allocated object \*/*  
                  )

**DESCRIPTION**    This function provides the default implementation of operator delete. It returns the memory, previously allocated by operator new, to the VxWorks system memory partition.

**RETURNS**        N/A

**SEE ALSO**        **cplusLib**

---

## operator new( )

**NAME** operator new( ) – default run-time support for operator new (C++)

**SYNOPSIS**

```
extern void * operator new
(
    size_t n                /* size of object to allocate */
)
```

**DESCRIPTION** This function provides the default implementation of operator new. It allocates memory from the system memory partition for the requested object. The value, when evaluated, is a pointer of the type **pointer-to-*T*** where *T* is the type of the new object.

If allocation fails a new-handler, if one is defined, is called. If the new-handler returns, presumably after attempting to recover from the memory allocation failure, allocation is retried. If there is no new-handler an exception of type “**bad\_alloc**” is thrown.

**RETURNS** Pointer to new object.

**THROWS** **std::bad\_alloc** if allocation failed.

**SEE ALSO** **cplusLib**

---

## operator new( )

**NAME** operator new( ) – default run-time support for operator new (nothrow) (C++)

**SYNOPSIS**

```
extern void * operator new
(
    size_t n,                /* size of object to allocate */
    const nothrow_t &       /* supply argument of "nothrow" here */
)
```

**DESCRIPTION** This function provides the default implementation of operator new (nothrow). It allocates memory from the system memory partition for the requested object. The value, when evaluated, is a pointer of the type **pointer-to-*T*** where *T* is the type of the new object.

If allocation fails, a new-handler, if one is defined, is called. If the new-handler returns, presumably after attempting to recover from the memory allocation failure, allocation is retried. If the **new\_handler** throws a **bad\_alloc** exception, the exception is caught and 0 is returned. If allocation fails and there is no **new\_handler** 0 is returned.

**RETURNS** Pointer to new object or 0 if allocation fails.

**INCLUDE FILES** new

**SEE ALSO** cplusplus

---

## operator new()

**NAME** operator new() – run-time support for operator new with placement (C++)

**SYNOPSIS**

```
extern void * operator new
(
    size_t n,                /* size of object to allocate (unused) */
    void * pMem              /* pointer to allocated memory */
)
```

**DESCRIPTION** This function provides the default implementation of the global new operator, with support for the placement syntax. New-with-placement is used to initialize objects for which memory has already been allocated. *pMem* points to the previously allocated memory.

**RETURNS** pMem

**INCLUDE FILES** new

**SEE ALSO** cplusplus

---

## passFsDevInit()

**NAME** `passFsDevInit()` – associate a device with passFs file system functions

**SYNOPSIS**

```
void *passFsDevInit
(
    char * devName          /* device name */
)
```

**DESCRIPTION** This routine associates the name *devName* with the file system and installs it in the I/O System's device table. The driver number used when the device is added to the table is that which was assigned to the passFs library during `passFsInit()`.

**RETURNS** A pointer to the volume descriptor, or NULL if there is an error.

**SEE ALSO** `passFsLib`

---

## passFsInit()

**NAME** `passFsInit()` – prepare to use the passFs library

**SYNOPSIS**

```
STATUS passFsInit
(
    int nPassfs             /* number of pass-through file systems */
)
```

**DESCRIPTION** This routine initializes the passFs library. It must be called exactly once, before any other routines in the library. The argument specifies the number of passFs devices that may be open at once. This routine installs `passFsLib` as a driver in the I/O system driver table, allocates and sets up the necessary memory structures, and initializes semaphores.

Normally this routine is called from the root task, `usrRoot()`, in `usrConfig()`. This initialization is enabled when the configuration macro `INCLUDE_PASSFS` is defined.

---

**NOTE:** Maximum number of pass-through file systems is 1.

---

**RETURNS** OK, or ERROR.

**SEE ALSO** `passFsLib`

**pause()**

---

## pause()

**NAME** `pause()` – suspend the task until delivery of a signal (POSIX)

**SYNOPSIS** `int pause (void)`

**DESCRIPTION** This routine suspends the task until delivery of a signal.

---

**NOTE:** Since the `pause()` function suspends thread execution indefinitely, there is no successful completion return value.

---

**RETURNS** -1, always.

**ERRNO** EINTR

**SEE ALSO** `sigLib`

---

## pc()

**NAME** `pc()` – return the contents of the program counter

**SYNOPSIS** `int pc  
(  
    int task                   /* task ID */  
)`

**DESCRIPTION** This command extracts the contents of the program counter for a specified task from the task's TCB. If `task` is omitted or 0, the current task is used.

**RETURNS** The contents of the program counter.

**SEE ALSO** `usrLib`, `ti()`, *VxWorks Programmer's Guide: Target Shell*

---

## pentiumBtc()

**NAME** pentiumBtc() – execute atomic compare-and-exchange instruction to clear a bit

**SYNOPSIS** `STATUS pentiumBtc (pFlag)  
char * pFlag; /* flag address */`

**DESCRIPTION** This routine compares a byte specified by the first parameter with **TRUE**. If it is **TRUE**, it changes it to 0 and returns **OK**. If it is not **TRUE**, it returns **ERROR**. **LOCK** and **CMPXCHGB** are used to get the atomic memory access.

**RETURNS** **OK** or **ERROR** if the specified flag is not **TRUE**

**SEE ALSO** pentiumALib

---

## pentiumBts()

**NAME** pentiumBts() – execute atomic compare-and-exchange instruction to set a bit

**SYNOPSIS** `STATUS pentiumBts (pFlag)  
char * pFlag; /* flag address */`

**DESCRIPTION** This routine compares a byte specified by the first parameter with 0. If it is 0, it changes it to **TRUE** and returns **OK**. If it is not 0, it returns **ERROR**. **LOCK** and **CMPXCHGB** are used to get the atomic memory access.

**RETURNS** **OK** or **ERROR** if the specified flag is not zero.

**SEE ALSO** pentiumALib

## **pentiumCr4Get()**

**NAME** `pentiumCr4Get()` – get contents of CR4 register

**SYNOPSIS** `int pentiumCr4Get (void)`

**DESCRIPTION** This routine gets the contents of the CR4 register.

**RETURNS** Contents of CR4 register.

**SEE ALSO** `pentiumALib`

---

## **pentiumCr4Set()**

**NAME** `pentiumCr4Set()` – sets specified value to the CR4 register

**SYNOPSIS** `void pentiumCr4Set (cr4)  
int cr4; /* value to write CR4 register */`

**DESCRIPTION** This routine sets a specified value to the CR4 register.

**RETURNS** N/A

**SEE ALSO** `pentiumALib`

---

## pentiumMcaEnable()

**NAME** `pentiumMcaEnable()` – enable/disable the MCA (Machine Check Architecture)

**SYNOPSIS**

```
void pentiumMcaEnable
(
    BOOL enable          /* TRUE to enable, FALSE to disable the MCA */
)
```

**DESCRIPTION** This routine enables/disables 1) the Machine Check Architecture and its Error Reporting register banks 2) the Machine Check Exception by toggling the MCE bit in the CR4. This routine works on either P5, P6 or P7 family.

**RETURNS** N/A

**SEE ALSO** `pentiumLib`

---

## pentiumMcaShow()

**NAME** `pentiumMcaShow()` – show MCA (Machine Check Architecture) registers

**SYNOPSIS**

```
void pentiumMcaShow (void)
```

**DESCRIPTION** This routine shows Machine-Check global control registers and Error-Reporting register banks. Number of the Error-Reporting register banks is kept in a variable `mcaBanks`. `MCI_ADDR` and `MCI_MISC` registers in the Error-Reporting register bank are showed if `MCI_STATUS` indicates that these registers are valid.

**RETURNS** N/A

**SEE ALSO** `pentiumShow`

---

## pentiumMsrGet()

**NAME** `pentiumMsrGet()` – get the contents of the specified MSR (Model Specific Register)

**SYNOPSIS**

```
void pentiumMsrGet (addr, pData)
    int addr;          /* MSR address */
    long long int * pData; /* MSR data */
```

**DESCRIPTION** This routine gets the contents of the specified MSR. The first parameter is an address of the MSR. The second parameter is a pointer of 64Bit variable.

**RETURNS** N/A

**SEE ALSO** `pentiumALib`

---

## pentiumMsrInit()

**NAME** `pentiumMsrInit()` – initialize all the MSRs (Model Specific Register)

**SYNOPSIS**

```
STATUS pentiumMsrInit (void)
```

**DESCRIPTION** This routine initializes all the MSRs in the processor. This routine works on either P5, P6 or P7 family processors.

**RETURNS** OK, or ERROR if RDMSR/WRMSR instructions are not supported.

**SEE ALSO** `pentiumLib`

---

## pentiumMsrSet()

**NAME** `pentiumMsrSet()` – set a value to the specified MSR (Model Specific Registers)

**SYNOPSIS**

```
void pentiumMsrSet (addr, pData)
    int addr;                /* MSR address */
    long long int * pData;   /* MSR data */
```

**DESCRIPTION** This routine sets a value to a specified MSR. The first parameter is an address of the MSR. The second parameter is a pointer of 64Bit variable.

**RETURNS** N/A

**SEE ALSO** `pentiumALib`

---

## pentiumMsrShow()

**NAME** `pentiumMsrShow()` – show all the MSR (Model Specific Register)

**SYNOPSIS**

```
void pentiumMsrShow (void)
```

**DESCRIPTION** This routine shows all the MSRs in the Pentium and Pentium[234].

**RETURNS** N/A

**SEE ALSO** `pentiumShow`

## **pentiumMtrrDisable()**

<b>NAME</b>	<b>pentiumMtrrDisable()</b> – disable MTRR (Memory Type Range Register)
<b>SYNOPSIS</b>	<code>void pentiumMtrrDisable (void)</code>
<b>DESCRIPTION</b>	This routine disables the MTRR that provide a mechanism for associating the memory types with physical address ranges in system memory.
<b>RETURNS</b>	N/A
<b>SEE ALSO</b>	<b>pentiumLib</b>

---

## **pentiumMtrrEnable()**

<b>NAME</b>	<b>pentiumMtrrEnable()</b> – enable MTRR (Memory Type Range Register)
<b>SYNOPSIS</b>	<code>void pentiumMtrrEnable (void)</code>
<b>DESCRIPTION</b>	This routine enables the MTRR that provide a mechanism for associating the memory types with physical address ranges in system memory.
<b>RETURNS</b>	N/A
<b>SEE ALSO</b>	<b>pentiumLib</b>

---

## pentiumMtrrGet()

**NAME** pentiumMtrrGet() – get MTRRs to a specified MTRR table

**SYNOPSIS**

```
STATUS pentiumMtrrGet
(
    MTRR * pMtrr          /* MTRR table */
)
```

**DESCRIPTION** This routine gets MTRRs to a specified MTRR table with RDMSR instruction. The read MTRRs are CAP register, DEFTYPE register, fixed range MTRRs, and variable range MTRRs.

**RETURNS** OK, or ERROR if MTRR is being accessed.

**SEE ALSO** pentiumLib

---

## pentiumMtrrSet()

**NAME** pentiumMtrrSet() – set MTRRs from specified MTRR table with WRMSR instruction.

**SYNOPSIS**

```
STATUS pentiumMtrrSet
(
    MTRR * pMtrr          /* MTRR table */
)
```

**DESCRIPTION** This routine sets MTRRs from specified MTRR table with WRMSR instruction. The written MTRRs are DEFTYPE register, fixed range MTRRs, and variable range MTRRs.

**RETURNS** OK, or ERROR if MTRR is enabled or being accessed.

**SEE ALSO** pentiumLib

---

## pentiumP5PmcGet()

<b>NAME</b>	<b>pentiumP5PmcGet()</b> – get the contents of P5 PMC0 and PMC1
<b>SYNOPSIS</b>	<pre>void pentiumP5PmcGet (pPmc0, pPmc1)     long long int * pPmc0;          /* Performance Monitoring Counter 0 */     long long int * pPmc1;          /* Performance Monitoring Counter 1 */</pre>
<b>DESCRIPTION</b>	This routine gets the contents of both PMC0 (Performance Monitoring Counter 0) and PMC1. The first parameter is a pointer of 64Bit variable to store the content of the Counter 0, and the second parameter is for the Counter 1.
<b>RETURNS</b>	N/A
<b>SEE ALSO</b>	<b>pentiumALib</b>

---

## pentiumP5PmcGet0()

<b>NAME</b>	<b>pentiumP5PmcGet0()</b> – get the contents of P5 PMC0
<b>SYNOPSIS</b>	<pre>void pentiumP5PmcGet0 (pPmc0)     long long int * pPmc0;          /* Performance Monitoring Counter 0 */</pre>
<b>DESCRIPTION</b>	This routine gets the contents of PMC0 (Performance Monitoring Counter 0). The parameter is a pointer of 64Bit variable to store the content of the Counter.
<b>RETURNS</b>	N/A
<b>SEE ALSO</b>	<b>pentiumALib</b>

---

## pentiumP5PmcGet1()

<b>NAME</b>	<b>pentiumP5PmcGet1()</b> – get the contents of P5 PMC1
<b>SYNOPSIS</b>	<pre>void pentiumP5PmcGet1 (pPmc1)     long long int * pPmc1;          /* Performance Monitoring Counter 1 */</pre>
<b>DESCRIPTION</b>	This routine gets a content of PMC1 (Performance Monitoring Counter 1). Parameter is a pointer of 64Bit variable to store the content of the Counter.
<b>RETURNS</b>	N/A
<b>SEE ALSO</b>	<b>pentiumALib</b>

---

## pentiumP5PmcReset()

<b>NAME</b>	<b>pentiumP5PmcReset()</b> – reset both PMC0 and PMC1
<b>SYNOPSIS</b>	<pre>void pentiumP5PmcReset (void)</pre>
<b>DESCRIPTION</b>	This routine resets both PMC0 (Performance Monitoring Counter 0) and PMC1.
<b>RETURNS</b>	N/A
<b>SEE ALSO</b>	<b>pentiumALib</b>

## **pentiumP5PmcReset0()**

**NAME**            **pentiumP5PmcReset0()** – reset PMC0

**SYNOPSIS**        **void pentiumP5PmcReset0 (void)**

**DESCRIPTION**     This routine resets PMC0 (Performance Monitoring Counter 0).

**RETURNS**        N/A

**SEE ALSO**        **pentiumALib**

---

## **pentiumP5PmcReset1()**

**NAME**            **pentiumP5PmcReset1()** – reset PMC1

**SYNOPSIS**        **void pentiumP5PmcReset1 (void)**

**DESCRIPTION**     This routine resets PMC1 (Performance Monitoring Counter 1).

**RETURNS**        N/A

**SEE ALSO**        **pentiumALib**

---

## pentiumP5PmcStart0( )

**NAME** pentiumP5PmcStart0( ) – start PMC0

**SYNOPSIS**

```
STATUS pentiumP5PmcStart0 (pmc0Cesr)
    int pmc0Cesr;          /* PMC0 control and event select */
```

**DESCRIPTION** This routine starts PMC0 (Performance Monitoring Counter 0) by writing specified PMC0 events to Performance Event Select Registers. The only parameter is the content of Performance Event Select Register.

**RETURNS** OK or ERROR if PMC0 is already started.

**SEE ALSO** pentiumALib

---

## pentiumP5PmcStart1( )

**NAME** pentiumP5PmcStart1( ) – start PMC1

**SYNOPSIS**

```
STATUS pentiumP5PmcStart1 (pmc1Cesr)
    int pmc1Cesr;          /* PMC1 control and event select */
```

**DESCRIPTION** This routine starts PMC1 (Performance Monitoring Counter 0) by writing specified PMC1 events to Performance Event Select Registers. The only parameter is the content of Performance Event Select Register.

**RETURNS** OK or ERROR if PMC1 is already started.

**SEE ALSO** pentiumALib

## **pentiumP5PmcStop()**

<b>NAME</b>	<b>pentiumP5PmcStop()</b> – stop both P5 PMC0 and PMC1
<b>SYNOPSIS</b>	<code>void pentiumP5PmcStop (void)</code>
<b>DESCRIPTION</b>	This routine stops both PMC0 (Performance Monitoring Counter 0) and PMC1 by clearing two Performance Event Select Registers.
<b>RETURNS</b>	N/A
<b>SEE ALSO</b>	<b>pentiumALib</b>

---

## **pentiumP5PmcStop0()**

<b>NAME</b>	<b>pentiumP5PmcStop0()</b> – stop P5 PMC0
<b>SYNOPSIS</b>	<code>void pentiumP5PmcStop0 (void)</code>
<b>DESCRIPTION</b>	This routine stops only PMC0 (Performance Monitoring Counter 0) by clearing the PMC0 bits of Control and Event Select Register.
<b>RETURNS</b>	N/A
<b>SEE ALSO</b>	<b>pentiumALib</b>

---

## pentiumP5PmcStop1()

<b>NAME</b>	<b>pentiumP5PmcStop1()</b> – stop P5 PMC1
<b>SYNOPSIS</b>	<code>void pentiumP5PmcStop1 (void)</code>
<b>DESCRIPTION</b>	This routine stops only PMC1 (Performance Monitoring Counter 1) by clearing the PMC1 bits of Control and Event Select Register.
<b>RETURNS</b>	N/A
<b>SEE ALSO</b>	<b>pentiumALib</b>

---

## pentiumP6PmcGet()

<b>NAME</b>	<b>pentiumP6PmcGet()</b> – get the contents of PMC0 and PMC1
<b>SYNOPSIS</b>	<pre>void pentiumP6PmcGet (pPmc0, pPmc1)     long long int * pPmc0;          /* Performance Monitoring Counter 0 */     long long int * pPmc1;          /* Performance Monitoring Counter 1 */</pre>
<b>DESCRIPTION</b>	This routine gets the contents of both PMC0 (Performance Monitoring Counter 0) and PMC1. The first parameter is a pointer of 64Bit variable to store the content of the Counter 0, and the second parameter is for the Counter 1.
<b>RETURNS</b>	N/A
<b>SEE ALSO</b>	<b>pentiumALib</b>

## **pentiumP6PmcGet0()**

<b>NAME</b>	<b>pentiumP6PmcGet0()</b> – get the contents of PMC0
<b>SYNOPSIS</b>	<pre>void pentiumP6PmcGet0 (pPmc0)     long long int * pPmc0;          /* Performance Monitoring Counter 0 */</pre>
<b>DESCRIPTION</b>	This routine gets the contents of PMC0 (Performance Monitoring Counter 0). The parameter is a pointer of 64Bit variable to store the content of the Counter.
<b>RETURNS</b>	N/A
<b>SEE ALSO</b>	<b>pentiumALib</b>

---

## **pentiumP6PmcGet1()**

<b>NAME</b>	<b>pentiumP6PmcGet1()</b> – get the contents of PMC1
<b>SYNOPSIS</b>	<pre>void pentiumP6PmcGet1 (pPmc1)     long long int * pPmc1;          /* Performance Monitoring Counter 1 */</pre>
<b>DESCRIPTION</b>	This routine gets a content of PMC1 (Performance Monitoring Counter 1). Parameter is a pointer of 64Bit variable to store the content of the Counter.
<b>RETURNS</b>	N/A
<b>SEE ALSO</b>	<b>pentiumALib</b>

---

## pentiumP6PmcReset()

**NAME** `pentiumP6PmcReset()` – reset both PMC0 and PMC1

**SYNOPSIS** `void pentiumP6PmcReset (void)`

**DESCRIPTION** This routine resets both PMC0 (Performance Monitoring Counter 0) and PMC1.

**RETURNS** N/A

**SEE ALSO** `pentiumALib`

---

## pentiumP6PmcReset0()

**NAME** `pentiumP6PmcReset0()` – reset PMC0

**SYNOPSIS** `void pentiumP6PmcReset0 (void)`

**DESCRIPTION** This routine resets PMC0 (Performance Monitoring Counter 0).

**RETURNS** N/A

**SEE ALSO** `pentiumALib`

---

## pentiumP6PmcReset1()

**NAME** `pentiumP6PmcReset1()` – reset PMC1

**SYNOPSIS** `void pentiumP6PmcReset1 (void)`

**DESCRIPTION** This routine resets PMC1 (Performance Monitoring Counter 1).

**RETURNS** N/A

**SEE ALSO** `pentiumALib`

---

## pentiumP6PmcStart()

**NAME** pentiumP6PmcStart() – start both PMC0 and PMC1

**SYNOPSIS**

```
STATUS pentiumP6PmcStart (pmcEvtSel0, pmcEvtSel1)
    int pmcEvtSel0;          /* Performance Event Select Register 0 */
    int pmcEvtSel1;          /* Performance Event Select Register 1 */
```

**DESCRIPTION** This routine starts both PMC0 (Performance Monitoring Counter 0) and PMC1 by writing specified events to Performance Event Select Registers. The first parameter is a content of Performance Event Select Register 0, and the second parameter is for the Performance Event Select Register 1.

**RETURNS** OK or ERROR if PMC is already started.

**SEE ALSO** pentiumALib

---

## pentiumP6PmcStop()

**NAME** pentiumP6PmcStop() – stop both PMC0 and PMC1

**SYNOPSIS**

```
void pentiumP6PmcStop (void)
```

**DESCRIPTION** This routine stops both PMC0 (Performance Monitoring Counter 0) and PMC1 by clearing two Performance Event Select Registers.

**RETURNS** N/A

**SEE ALSO** pentiumALib

---

## pentiumP6PmcStop1()

**NAME** `pentiumP6PmcStop1()` – stop PMC1

**SYNOPSIS** `void pentiumP6PmcStop1 (void)`

**DESCRIPTION** This routine stops only PMC1 (Performance Monitoring Counter 1) by clearing the Performance Event Select Register 1. Note, clearing the Performance Event Select Register 0 stops both counters, PMC0 and PMC1.

**RETURNS** N/A

**SEE ALSO** `pentiumALib`

---

## pentiumPmcGet()

**NAME** `pentiumPmcGet()` – get the contents of PMC0 and PMC1

**SYNOPSIS**

```
void pentiumPmcGet (pPmc0, pPmc1)
    long long int * pPmc0;          /* Performance Monitoring Counter 0 */
    long long int * pPmc1;          /* Performance Monitoring Counter 1 */
```

**DESCRIPTION** This routine gets the contents of both PMC0 (Performance Monitoring Counter 0) and PMC1. The first parameter is a pointer of 64Bit variable to store the content of the Counter 0, and the second parameter is for the Counter 1.

**RETURNS** N/A

**SEE ALSO** `pentiumLib`

---

## pentiumPmcGet0()

<b>NAME</b>	<b>pentiumPmcGet0()</b> – get the contents of PMC0
<b>SYNOPSIS</b>	<pre>void pentiumPmcGet0 (pPmc0)     long long int * pPmc0;          /* Performance Monitoring Counter 0 */</pre>
<b>DESCRIPTION</b>	This routine gets the contents of PMC0 (Performance Monitoring Counter 0). The parameter is a pointer of 64Bit variable to store the content of the Counter.
<b>RETURNS</b>	N/A
<b>SEE ALSO</b>	<b>pentiumLib</b>

---

## pentiumPmcGet1()

<b>NAME</b>	<b>pentiumPmcGet1()</b> – get the contents of PMC1
<b>SYNOPSIS</b>	<pre>void pentiumPmcGet1 (pPmc1)     long long int * pPmc1;          /* Performance Monitoring Counter 1 */</pre>
<b>DESCRIPTION</b>	This routine gets a content of PMC1 (Performance Monitoring Counter 1). Parameter is a pointer of 64Bit variable to store the content of the Counter.
<b>RETURNS</b>	N/A
<b>SEE ALSO</b>	<b>pentiumLib</b>

---

## pentiumPmcReset()

**NAME** `pentiumPmcReset()` – reset both PMC0 and PMC1

**SYNOPSIS** `void pentiumPmcReset (void)`

**DESCRIPTION** This routine resets both PMC0 (Performance Monitoring Counter 0) and PMC1.

**RETURNS** N/A

**SEE ALSO** `pentiumLib`

---

## pentiumPmcReset0()

**NAME** `pentiumPmcReset0()` – reset PMC0

**SYNOPSIS** `void pentiumPmcReset0 (void)`

**DESCRIPTION** This routine resets PMC0 (Performance Monitoring Counter 0).

**RETURNS** N/A

**SEE ALSO** `pentiumLib`

---

## pentiumPmcReset1()

**NAME** `pentiumPmcReset1()` – reset PMC1

**SYNOPSIS** `void pentiumPmcReset1 (void)`

**DESCRIPTION** This routine resets PMC1 (Performance Monitoring Counter 1).

**RETURNS** N/A

**SEE ALSO** `pentiumLib`

---

## pentiumPmcShow()

**NAME** pentiumPmcShow() – show PMCs (Performance Monitoring Counters)

**SYNOPSIS**

```
void pentiumPmcShow
(
    BOOL zap                /* 1: reset PMC0 and PMC1 */
)
```

**DESCRIPTION** This routine shows Performance Monitoring Counter 0 and 1. Monitored events are selected by Performance Event Select Registers in **pentiumPmcStart()**. These counters are cleared to 0 if the parameter “zap” is TRUE.

**RETURNS** N/A

**SEE ALSO** pentiumShow

---

## pentiumPmcStart()

**NAME** pentiumPmcStart() – start both PMC0 and PMC1

**SYNOPSIS**

```
STATUS pentiumPmcStart (pmcEvtSel0, pmcEvtSel1)
    int pmcEvtSel0;        /* Performance Event Select Register 0 */
    int pmcEvtSel1;        /* Performance Event Select Register 1 */
```

**DESCRIPTION** This routine starts both PMC0 (Performance Monitoring Counter 0) and PMC1 by writing specified events to Performance Event Select Registers. The first parameter is a content of Performance Event Select Register 0, and the second parameter is for the Performance Event Select Register 1.

**RETURNS** OK or ERROR if PMC is already started.

**SEE ALSO** pentiumLib

---

## pentiumPmcStart0( )

**NAME** pentiumPmcStart0( ) – start PMC0

**SYNOPSIS** `STATUS pentiumPmcStart0 (pmcEvtSel0)  
int pmcEvtSel0; /* PMC0 control and event select */`

This routine starts PMC0 (Performance Monitoring Counter 0) by writing specified PMC0 events to Performance Event Select Registers. The only parameter is the content of Performance Event Select Register.

**RETURNS** OK or ERROR if PMC is already started.

**SEE ALSO** pentiumLib

---

## pentiumPmcStart1( )

**NAME** pentiumPmcStart1( ) – start PMC1

**SYNOPSIS** `STATUS pentiumPmcStart1 (pmcEvtSel1)  
int pmcEvtSel1; /* PMC1 control and event select */`

This routine starts PMC1 (Performance Monitoring Counter 0) by writing specified PMC1 events to Performance Event Select Registers. The only parameter is the content of Performance Event Select Register.

**RETURNS** OK or ERROR if PMC1 is already started.

**SEE ALSO** pentiumLib

## **pentiumPmcStop()**

**NAME** `pentiumPmcStop()` – stop both PMC0 and PMC1

**SYNOPSIS** `void pentiumPmcStop (void)`

This routine stops both PMC0 (Performance Monitoring Counter 0) and PMC1 by clearing two Performance Event Select Registers.

**RETURNS** N/A

**SEE ALSO** `pentiumLib`

---

## **pentiumPmcStop0()**

**NAME** `pentiumPmcStop0()` – stop PMC0

**SYNOPSIS** `void pentiumPmcStop0 (void)`

This routine stops only PMC0 (Performance Monitoring Counter 0) by clearing the PMC0 bits of Control and Event Select Register.

**RETURNS** N/A

**SEE ALSO** `pentiumLib`

---

## **pentiumPmcStop1()**

**NAME** `pentiumPmcStop1()` – stop PMC1

**SYNOPSIS** `void pentiumPmcStop1 (void)`

This routine stops only PMC1 (Performance Monitoring Counter 1) by clearing the PMC1 bits of Control and Event Select Register.

**RETURNS** N/A

**SEE ALSO** `pentiumLib`

---

## pentiumSerialize()

<b>NAME</b>	<b>pentiumSerialize()</b> – execute a serializing instruction CPUID
<b>SYNOPSIS</b>	<b>void pentiumSerialize (void)</b>
<b>DESCRIPTION</b>	This routine executes a serializing instruction CPUID. Serialization means that all modifications to flags, registers, and memory by previous instructions are completed before the next instruction is fetched and executed and all buffered writes have drained to memory.
<b>RETURNS</b>	N/A
<b>SEE ALSO</b>	<b>pentiumALib</b>

---

## pentiumTlbFlush()

<b>NAME</b>	<b>pentiumTlbFlush()</b> – flush TLBs (Translation Lookaside Buffers)
<b>SYNOPSIS</b>	<b>void pentiumTlbFlush (void)</b>
<b>DESCRIPTION</b>	This routine flushes TLBs by loading the CR3 register. All of the TLBs are automatically invalidated any time the CR3 register is loaded. The page global enable (PGE) flag in register CR4 and the global flag in a page-directory or page-table entry can be used to frequently used pages from being automatically invalidated in the TLBs on a load of CR3 register. The only way to deterministically invalidate global page entries is to clear the PGE flag and then invalidate the TLBs.
<b>RETURNS</b>	N/A
<b>SEE ALSO</b>	<b>pentiumALib</b>

## **pentiumTscGet32()**

**NAME** `pentiumTscGet32()` – get the lower half of the 64Bit TSC (Timestamp Counter)

**SYNOPSIS** `UINT32 pentiumTscGet32 (void)`

**DESCRIPTION** This routine gets a lower half of the 64Bit TSC by RDTSC instruction. RDTSC instruction saves the lower 32Bit in EAX register, so this routine simply returns after executing RDTSC instruction.

**RETURNS** Lower half of the 64Bit TSC (Timestamp Counter)

**SEE ALSO** `pentiumALib`

---

## **pentiumTscGet64()**

**NAME** `pentiumTscGet64()` – get 64Bit TSC (Timestamp Counter)

**SYNOPSIS** `void pentiumTscGet64 (pTsc)  
          long long int * pTsc;                  /* Timestamp Counter */`

**DESCRIPTION** This routine gets 64Bit TSC by RDTSC instruction. Parameter is a pointer of 64Bit variable to store the content of the Counter.

**RETURNS** N/A

**SEE ALSO** `pentiumALib`

---

## **pentiumTscReset()**

**NAME** `pentiumTscReset()` – reset the TSC (Timestamp Counter)

**SYNOPSIS** `void pentiumTscReset (void)`

**DESCRIPTION** This routine resets the TSC by writing zero to the TSC with WRMSR instruction.

**RETURNS** N/A

**SEE ALSO** `pentiumALib`

---

## period()

**NAME** `period()` – spawn a task to call a function periodically

**SYNOPSIS**

```
int period
(
    int     secs,           /* period in seconds */
    FUNCPTR func,         /* function to call repeatedly */
    int     arg1,          /* first of eight args to pass to func */
    int     arg2,
    int     arg3,
    int     arg4,
    int     arg5,
    int     arg6,
    int     arg7,
    int     arg8
)
```

**DESCRIPTION** This command spawns a task that repeatedly calls a specified function, with up to eight of its arguments, delaying the specified number of seconds between calls.

For example, to have `i()` display task information every 5 seconds, just type:

```
-> period 5, i
```

---

**NOTE:** The task is spawned using the `sp()` routine. See the description of `sp()` for details about priority, options, stack size, and task ID.

---

**RETURNS** A task ID, or **ERROR** if the task cannot be spawned.

**SEE ALSO** `usrLib`, `periodRun()`, `sp()`, *VxWorks Programmer's Guide: Target Shell*, `windsh`, *Tornado User's Guide: Shell*

## periodRun()

**NAME** `periodRun()` – call a function periodically

**SYNOPSIS**

```
void periodRun
(
    int      secs,           /* no. of seconds to delay between calls */
    FUNCPTR  func,         /* function to call repeatedly */
    int      arg1,          /* first of eight args to pass to func */
    int      arg2,
    int      arg3,
    int      arg4,
    int      arg5,
    int      arg6,
    int      arg7,
    int      arg8
)
```

**DESCRIPTION** This command repeatedly calls a specified function, with up to eight of its arguments, delaying the specified number of seconds between calls.

Normally, this routine is called only by `period()`, which spawns it as a task.

**RETURNS** N/A

**SEE ALSO** `usrLib`, `period()`, *VxWorks Programmer's Guide: Target Shell*

---

## perror()

**NAME** `perror()` – map an error number in `errno` to an error message (ANSI)

**SYNOPSIS**

```
void perror
(
    const char * __s      /* error string */
)
```

**DESCRIPTION** This routine maps the error number in the integer expression `errno` to an error message. It writes a sequence of characters to the standard error stream as follows: first (if `__s` is not a null pointer and the character pointed to by `__s` is not the null character), the string pointed to by `__s` followed by a colon (:) and a space; then an appropriate error message

string followed by a new-line character. The contents of the error message strings are the same as those returned by **strerror()** with the argument **errno**.

<b>INCLUDE FILES</b>	<b>stdio.h</b>
<b>RETURNS</b>	N/A
<b>SEE ALSO</b>	<b>ansiStdio, strerror()</b>

---

## ping()

**NAME** ping() – test that a remote host is reachable

**SYNOPSIS**

```
STATUS ping
(
    char * host,           /* host to ping */
    int numPackets,       /* number of packets to receive */
    ulong_t options       /* option flags */
)
```

**DESCRIPTION** This routine tests that a remote host is reachable by sending ICMP echo request packets, and waiting for replies. It may be called from the VxWorks shell as follows:

```
-> ping "remoteSystem", 1, 0
```

where *remoteSystem* is either a host name that has been previously added to the remote host table by a call to **hostAdd()**, or an Internet address in dot notation (for example, "90.0.0.2").

The second parameter, *numPackets*, specifies the number of ICMP packets to receive from the remote host. If *numPackets* is 1, this routine waits for a single echo reply packet, and then prints a short message indicating whether the remote host is reachable. For all other values of *numPackets*, timing and sequence information is printed as echoed packets are received. If *numPackets* is 0, this routine runs continuously.

If no replies are received within a 5-second timeout period, the routine exits. An **ERROR** status is returned if no echo replies are received from the remote host.

The following flags may be given through the *options* parameter:

**PING\_OPT\_SILENT**

Suppress output. This option is useful for applications that use **ping()** programmatically to examine the return status.

**PING\_OPT\_DONTROUTE**

Do not route packets past the local network. This also prevents pinging local addresses (*i.e.*, the IP address of the host itself). The 127.x.x.x addresses will still work however.

**PING\_OPT\_NOHOST**

Suppress host lookup. This is useful when you have the DNS resolver but the DNS server is down and not returning host names.

**PING\_OPT\_DEBUG**

Enables debug output.

**NOTE** The following global variables can be set from the target shell or **Windsh** to configure the **ping()** parameters:

- `_pingTxLen`  
Size of the ICMP echo packet (default 64).
- `_pingTxInterval`  
Packet interval in seconds (default 1 second).
- `_pingTxTmo`  
Packet timeout in seconds (default 5 seconds).

**RETURNS** OK, or ERROR if the remote host is not reachable.

**ERRNO** EINVAL, S\_pingLib\_NOT\_INITIALIZED, S\_pingLib\_TIMEOUT

**SEE ALSO** pingLib

---

## pingLibInit()

**NAME** pingLibInit() – initialize the ping() utility

**SYNOPSIS** STATUS pingLibInit (void)

**DESCRIPTION** This routine allocates resources used by the ping() utility. It is called automatically when INCLUDE\_PING is defined.

**RETURNS** OK

**SEE ALSO** pingLib

---

## pipeDevCreate()

**NAME** pipeDevCreate() – create a pipe device

**SYNOPSIS**

```
STATUS pipeDevCreate
(
    char * name,           /* name of pipe to be created */
    int   nMessages,      /* max. number of messages in pipe */
    int   nBytes          /* size of each message */
)
```

**DESCRIPTION** This routine creates a pipe device. It cannot be called from an interrupt service routine. It allocates memory for the necessary structures and initializes the device. The pipe device will have a maximum of *nMessages* messages of up to *nBytes* each in the pipe at once. When the pipe is full, a task attempting to write to the pipe will be suspended until a message has been read. Messages are lost if written to a full pipe at interrupt level.

**RETURNS** OK, or ERROR if the call fails.

**ERRNO** S\_ioLib\_NO\_DRIVER - driver not initialized  
S\_intLib\_NOT\_ISR\_CALLABLE - cannot be called from an ISR

**SEE ALSO** pipeDrv

---

## pipeDevDelete()

**NAME** pipeDevDelete() – delete a pipe device

**SYNOPSIS**

```
STATUS pipeDevDelete
(
    char * name,           /* name of pipe to be deleted */
    BOOL  force           /* if TRUE, force pipe deletion */
)
```

**DESCRIPTION** This routine deletes a pipe device of a given name. The name must match that passed to **pipeDevCreate()** else **ERROR** will be returned. This routine frees memory for the necessary structures and deletes the device. It cannot be called from an interrupt service routine.

**pipeDrv()**

A pipe device cannot be deleted until its number of open requests has been reduced to zero by an equal number of close requests and there are no tasks pending in its select list. If the optional force flag is asserted, the above restrictions are ignored, resulting in forced deletion of any select list and freeing of pipe resources.

---

**WARNING:** Forced pipe deletion can have catastrophic results if used indiscriminately. Use only as a last resort.

---

<b>RETURNS</b>	OK, or <b>ERROR</b> if the call fails.
<b>ERRNO</b>	<b>S_ioLib_NO_DRIVER</b> - driver not initialized <b>S_intLib_NOT_ISR_CALLABLE</b> - cannot be called from an ISR <b>EMFILE</b> - pipe still has other openings <b>EBUSY</b> - pipe is selected by at least one pending task
<b>SEE ALSO</b>	<b>pipeDrv</b>

---

## pipeDrv()

<b>NAME</b>	<b>pipeDrv()</b> – initialize the pipe driver
<b>SYNOPSIS</b>	<b>STATUS pipeDrv (void)</b>
<b>DESCRIPTION</b>	This routine initializes and installs the driver. It must be called before any pipes are created. It is called automatically by the root task, <b>usrRoot()</b> , in <b>usrConfig.c</b> when the configuration macro <b>INCLUDE_PIPES</b> is defined.
<b>RETURNS</b>	OK, or <b>ERROR</b> if the driver installation fails.
<b>SEE ALSO</b>	<b>pipeDrv</b>

---

## pow()

**NAME** `pow()` – compute the value of a number raised to a specified power (ANSI)

**SYNOPSIS**

```
double pow
(
    double x,          /* operand */
    double y          /* exponent */
)
```

**DESCRIPTION** This routine returns  $x$  to the power of  $y$  in double precision (IEEE double, 53 bits). A domain error occurs if  $x$  is negative and  $y$  is not an integral value. A domain error occurs if the result cannot be represented when  $x$  is zero and  $y$  is less than or equal to zero. A range error may occur.

**INCLUDE FILES** `math.h`

**RETURNS** The double-precision value of  $x$  to the power of  $y$ . Special cases:

<code>(anything) ** 0</code>	is	1
<code>(anything) ** 1</code>	is	itself
<code>(anything) ** NaN</code>	is	NaN
<code>NaN ** (anything except 0)</code>	is	NaN
<code>+-(anything &gt; 1) ** +INF</code>	is	+INF
<code>+-(anything &gt; 1) ** -INF</code>	is	+0
<code>+-(anything &lt; 1) ** +INF</code>	is	+0
<code>+-(anything &lt; 1) ** -INF</code>	is	+INF
<code>+1 ** +-INF</code>	is	NaN, signal INVALID
<code>+0 ** +(anything non-0, NaN)</code>	is	+0
<code>-0 ** +(anything non-0, NaN, odd int)</code>	is	+0
<code>+0 ** -(anything non-0, NaN)</code>	is	+INF, signal DIV-BY-ZERO
<code>-0 ** -(anything non-0, NaN, odd int)</code>	is	+INF with signal
<code>-0 ** (odd integer)</code>	=	-(+0 ** (odd integer))
<code>+INF ** +(anything except 0, NaN)</code>	is	+INF
<code>+INF ** -(anything except 0, NaN)</code>	is	+0
<code>-INF ** (odd integer)</code>	=	-(+INF ** (odd integer))
<code>-INF ** (even integer)</code>	=	(+INF ** (even integer))
<code>-INF ** -(any non-integer, NaN)</code>	is	NaN with signal
<code>-(x=anything) ** (k=integer)</code>	is	(-1)**k * (x ** k)
<code>-(anything except 0) ** (non-integer)</code>	is	NaN with signal

**SEE ALSO** `ansiMath`, `mathALib`

**powf()**

---

**powf()**

**NAME** `powf()` – compute the value of a number raised to a specified power (ANSI)

**SYNOPSIS**

```
float powf
(
    float x,    /* operand */
    float y     /* exponent */
)
```

**DESCRIPTION** This routine returns the value of  $x$  to the power of  $y$  in single precision.

**INCLUDE FILES** `math.h`

**RETURNS** The single-precision value of  $x$  to the power of  $y$ .

**SEE ALSO** `mathALib`

---

**pppDelete()**

**NAME** `pppDelete()` – delete a PPP network interface

**SYNOPSIS**

```
void pppDelete
(
    int unit                /* PPP interface unit number to delete */
)
```

**DESCRIPTION** This routine deletes the Point-to-Point Protocol (PPP) network interface specified by the unit number *unit*.

A Link Control Protocol (LCP) terminate request packet is sent to notify the peer of the impending PPP link shut-down. The associated serial interface (*tty*) is then detached from the PPP driver, and the PPP interface is deleted from the list of network interfaces. Finally, all resources associated with the PPP link are returned to the VxWorks system.

**RETURNS** N/A

**SEE ALSO** `pppLib`

---

## pppHookAdd()

**NAME** `pppHookAdd()` – add a hook routine on a unit basis

**SYNOPSIS**

```
STATUS pppHookAdd
(
    int    unit,           /* unit number */
    FUNCPTR hookRtn,      /* hook routine */
    int    hookType       /* hook type connect/disconnect */
)
```

**DESCRIPTION** This routine adds a hook to the Point-to-Point Protocol (PPP) channel. The parameters to this routine specify the unit number (*unit*) of the PPP interface, the hook routine (*hookRtn*), and the type of hook specifying either a connect hook or a disconnect hook (*hookType*). The following hook types can be specified for the *hookType* parameter:

**PPP\_HOOK\_CONNECT**  
Specify a connect hook.

**PPP\_HOOK\_DISCONNECT**  
Specify a disconnect hook.

**RETURNS** OK, or ERROR if the hook cannot be added to the unit.

**SEE ALSO** `pppHookLib`, `pppHookDelete()`

**P**

---

## pppHookDelete()

**NAME** `pppHookDelete()` – delete a hook routine on a unit basis

**SYNOPSIS**

```
STATUS pppHookDelete
(
    int unit,           /* unit number */
    int hookType       /* hook type connect/disconnect */
)
```

**DESCRIPTION** This routine deletes a hook added previously to the Point-to-Point Protocol (PPP) channel. The parameters to this routine specify the unit number (*unit*) of the PPP interface and the type of hook specifying either a connect hook or a disconnect hook (*hookType*). The following hook types can be specified for the *hookType* parameter:

**PPP\_HOOK\_CONNECT**  
Specify a connect hook.

**PPP\_HOOK\_DISCONNECT**  
Specify a disconnect hook.

**RETURNS** OK, or **ERROR** if the hook cannot be deleted for the unit.

**SEE ALSO** **pppHookLib**, **pppHookAdd()**

---

## pppInfoGet()

**NAME** **pppInfoGet()** – get PPP link status information

**SYNOPSIS**

```
STATUS pppInfoGet
(
    int          unit,          /* PPP interface unit number to examine */
    PPP_INFO * pInfo          /* PPP_INFO structure to be filled */
)
```

**DESCRIPTION** This routine gets status information pertaining to the specified Point-to-Point Protocol (PPP) link, regardless of the link state. State and option information is gathered for the Link Control Protocol (LCP), Internet Protocol Control Protocol (IPCP), Password Authentication Protocol (PAP), and Challenge-Handshake Authentication Protocol (CHAP).

The PPP link information is returned through a **PPP\_INFO** structure, which is defined in **h/netinet/ppp/pppShow.h**.

**RETURNS** OK, or **ERROR** if *unit* is an invalid PPP unit number.

**SEE ALSO** **pppShow**, **pppLib**

---

## pppInfoShow()

<b>NAME</b>	<code>pppInfoShow()</code> – display PPP link status information
<b>SYNOPSIS</b>	<pre>void pppInfoShow (void)</pre>
<b>DESCRIPTION</b>	This routine displays status information pertaining to each initialized Point-to-Point Protocol (PPP) link, regardless of the link state. State and option information is gathered for the Link Control Protocol (LCP), Internet Protocol Control Protocol (IPCP), Password Authentication Protocol (PAP), and Challenge-Handshake Authentication Protocol (CHAP).
<b>RETURNS</b>	N/A
<b>SEE ALSO</b>	<code>pppShow</code> , <code>pppLib</code>

---

## pppInit()

<b>NAME</b>	<code>pppInit()</code> – initialize a PPP network interface
<b>SYNOPSIS</b>	<pre>int pppInit (     int          unit,          /* PPP interface unit number to initialize */     char *       devname,      /* name of the tty device to be used */     char *       local_addr,   /* local IP address of the PPP interface */     char *       remote_addr,  /* remote peer IP address of the PPP link */     int          baud,         /* baud rate of tty; NULL = default */     PPP_OPTIONS * pOptions,    /* PPP options structure pointer */     char *       fOptions      /* PPP options file name */ )</pre>
<b>DESCRIPTION</b>	This routine initializes a Point-to-Point Protocol (PPP) network interface. The parameters to this routine specify the unit number ( <i>unit</i> ) of the PPP interface, the name of the serial interface ( <i>tty</i> ) device ( <i>devname</i> ), the IP addresses of the local and remote ends of the link, the interface baud rate, an optional configuration options structure pointer, and an optional configuration options file name.
<b>IP ADDRESSES</b>	The <i>local_addr</i> and <i>remote_addr</i> parameters specify the IP addresses of the local and remote ends of the PPP link, respectively. If <i>local_addr</i> is <code>NULL</code> , the local IP address will be

negotiated with the remote peer. If the remote peer does not assign a local IP address, it will default to the address associated with the local target's machine name. If *remote\_addr* is **NULL**, the remote peer's IP address will be obtained from the remote peer. A routing table entry to the remote peer will be automatically added once the PPP link is established.

#### **CONFIGURATION OPTIONS STRUCTURE**

The optional parameter *pOptions* specifies configuration options for the PPP link. If **NULL**, this parameter is ignored, otherwise it is assumed to be a pointer to a **PPP\_OPTIONS** options structure (defined in **h/netinet/ppp/options.h**).

The "flags" member of the **PPP\_OPTIONS** structure is a bit-mask, where the following bit-flags may be specified:

**OPT\_NO\_ALL**

Do not request/allow any options.

**OPT\_PASSIVE\_MODE**

Set passive mode.

**OPT\_SILENT\_MODE**

Set silent mode.

**OPT\_DEFAULTROUTE**

Add default route.

**OPT\_PROXYARP**

Add proxy ARP entry.

**OPT\_IPCP\_ACCEPT\_LOCAL**

Accept peer's idea of the local IP address.

**OPT\_IPCP\_ACCEPT\_REMOTE**

Accept peer's idea of the remote IP address.

**OPT\_NO\_IP**

Disable IP address negotiation.

**OPT\_NO\_ACC**

Disable address/control compression.

**OPT\_NO\_PC**

Disable protocol field compression.

**OPT\_NO\_VJ**

Disable VJ (Van Jacobson) compression.

**OPT\_NO\_VJCCOMP**

Disable VJ (Van Jacobson) connection ID compression.

**OPT\_NO\_ASYNCMAP**

Disable async map negotiation.

**OPT\_NO\_MN**

Disable magic number negotiation.

**OPT\_NO\_MRU**  
Disable MRU (Maximum Receive Unit) negotiation.

**OPT\_NO\_PAP**  
Do not allow PAP authentication with peer.

**OPT\_NO\_CHAP**  
Do not allow CHAP authentication with peer.

**OPT\_REQUIRE\_PAP**  
Require PAP authentication with peer.

**OPT\_REQUIRE\_CHAP**  
Require CHAP authentication with peer.

**OPT\_LOGIN**  
Use the login password database for PAP authentication of peer.

**OPT\_DEBUG**  
Enable PPP daemon debug mode.

**OPT\_DRIVER\_DEBUG**  
Enable PPP driver debug mode.

The remaining members of the **PPP\_OPTIONS** structure specify PPP configurations options that require string values. These options are:

char \***asyncmap**  
Set the desired async map to the specified string.

char \***escape\_chars**  
Set the chars to escape on transmission to the specified string.

char \***vj\_max\_slots**  
Set maximum number of VJ compression header slots to the specified string.

char \***netmask**  
Set netmask value for negotiation to the specified string.

char \***mru**  
Set MRU value for negotiation to the specified string.

char \***mtu**  
Set MTU (Maximum Transmission Unit) value for negotiation to the specified string.

char \***lcp\_echo\_failure**  
Set the maximum number of consecutive LCP echo failures to the specified string.

char \***lcp\_echo\_interval**  
Set the interval in seconds between LCP echo requests to the specified string.

char \***lcp\_restart**  
Set the timeout in seconds for the LCP negotiation to the specified string.

**pppInit()**

char **\*lcp\_max\_terminate**

Set the maximum number of transmissions for LCP termination requests to the specified string.

char **\*lcp\_max\_configure**

Set the maximum number of transmissions for LCP configuration requests to the specified string.

char **\*lcp\_max\_failure**

Set the maximum number of LCP configuration NAKs to the specified string.

char **\*ipcp\_restart**

Set the timeout in seconds for IPCP negotiation to the specified string.

char **\*ipcp\_max\_terminate**

Set the maximum number of transmissions for IPCP termination requests to the specified string.

char **\*ipcp\_max\_configure**

Set the maximum number of transmissions for IPCP configuration requests to the specified string.

char **\*ipcp\_max\_failure**

Set the maximum number of IPCP configuration NAKs to the specified string.

char **\*local\_auth\_name**

Set the local name for authentication to the specified string.

char **\*remote\_auth\_name**

Set the remote name for authentication to the specified string.

char **\*pap\_file**

Get PAP secrets from the specified file. This option is necessary if either peer requires PAP authentication.

char **\*pap\_user\_name**

Set the user name for PAP authentication with the peer to the specified string.

char **\*pap\_passwd**

Set the password for PAP authentication with the peer to the specified string.

char **\*pap\_restart**

Set the timeout in seconds for PAP negotiation to the specified string.

char **\*pap\_max\_authreq**

Set the maximum number of transmissions for PAP authentication requests to the specified string.

char **\*chap\_file**

Get CHAP secrets from the specified file. This option is necessary if either peer requires CHAP authentication.

**char \*chap\_restart**  
Set the timeout in seconds for CHAP negotiation to the specified string.

**char \*chap\_interval**  
Set the interval in seconds for CHAP re-challenge to the specified string.

**char \*chap\_max\_challenge**  
Set the maximum number of transmissions for CHAP challenge to the specified string.

#### CONFIGURATION OPTIONS FILE

The optional parameter *fOptions* specifies configuration options for the PPP link. If NULL, this parameter is ignored, otherwise it is assumed to be the name of a configuration options file. The format of the options file is one option per line; comment lines start with "#". The following options are recognized:

**no\_all**  
Do not request/allow any options.

**passive\_mode**  
Set passive mode.

**silent\_mode**  
Set silent mode.

**defaultroute**  
Add default route.

**proxyarp**  
Add proxy ARP entry.

**ipcp\_accept\_local**  
Accept peer's idea of the local IP address.

**ipcp\_accept\_remote**  
Accept peer's idea of the remote IP address.

**no\_ip**  
Disable IP address negotiation.

**no\_acc**  
Disable address/control compression.

**no\_pc**  
Disable protocol field compression.

**no\_vj**  
Disable VJ (Van Jacobson) compression.

**no\_vjcomp**  
Disable VJ (Van Jacobson) connection ID compression.

**pppInit()**

no\_asyncmap

Disable async map negotiation.

no\_mn

Disable magic number negotiation.

no\_mru

Disable MRU (Maximum Receive Unit) negotiation.

no\_pap

Do not allow PAP authentication with peer.

no\_chap

Do not allow CHAP authentication with peer.

require\_pap

Require PAP authentication with peer.

require\_chap

Require CHAP authentication with peer.

login

Use the login password database for PAP authentication of peer.

debug

Enable PPP daemon debug mode.

driver\_debug

Enable PPP driver debug mode.

asyncmap *value*

Set the desired async map to the specified value.

**escape\_chars** *value*

Set the chars to escape on transmission to the specified value.

**vj\_max\_slots** *value*

Set maximum number of VJ compression header slots to the specified value.

netmask *value*

Set netmask value for negotiation to the specified value.

mru *value*

Set MRU value for negotiation to the specified value.

mtu *value*

Set MTU value for negotiation to the specified value.

**lcp\_echo\_failure** *value*

Set the maximum consecutive LCP echo failures to the specified value.

**lcp\_echo\_interval** *value*

Set the interval in seconds between LCP echo requests to the specified value.

**lcp\_restart** *value*

Set the timeout in seconds for the LCP negotiation to the specified value.

**lcp\_max\_terminate** *value*

Set the maximum number of transmissions for LCP termination requests.

**lcp\_max\_configure** *value*

Set the maximum number of transmissions for LCP configuration requests to the specified value.

**lcp\_max\_failure** *value*

Set the maximum number of LCP configuration NAKs to the specified value.

**ipcp\_restart** *value*

Set the timeout in seconds for IPCP negotiation to the specified value.

**ipcp\_max\_terminate** *value*

Set the maximum number of transmissions for IPCP termination requests to the specified value.

**ipcp\_max\_configure** *value*

Set the maximum number of transmissions for IPCP configuration requests to the specified value.

**ipcp\_max\_failure** *value*

Set the maximum number of IPCP configuration NAKs to the specified value.

**local\_auth\_name** *name*

Set the local name for authentication to the specified name.

**remote\_auth\_name** *name*

Set the remote name for authentication to the specified name.

**pap\_file** *file*

Get PAP secrets from the specified file. This option is necessary if either peer requires PAP authentication.

**pap\_user\_name** *name*

Set the user name for PAP authentication with the peer to the specified name.

-

Set the password for PAP authentication with the peer to the specified password.

**pap\_restart** *value*

Set the timeout in seconds for PAP negotiation to the specified value.

**pap\_max\_authreq** *value*

Set the maximum number of transmissions for PAP authentication requests to the specified value.

**chap\_file** *file*

Get CHAP secrets from the specified file. This option is necessary if either peer requires CHAP authentication.

**pppInit()****chap\_restart** *value*

Set the timeout in seconds for CHAP negotiation to the specified value.

**chap\_interval** *value*

Set the interval in seconds for CHAP re-challenge to the specified value.

**chap\_max\_challenge** *value*

Set the maximum number of transmissions for CHAP challenge to the specified value.

**AUTHENTICATION** The VxWorks PPP implementation supports two separate user authentication protocols: the Password Authentication Protocol (PAP) and the Challenge-Handshake Authentication Protocol (CHAP). If authentication is required by either peer, it must be satisfactorily completed before the PPP link becomes fully operational. If authentication fails, the link will be automatically terminated.

**EXAMPLES** The following routine initializes a PPP interface that uses the target's second serial port (*/tyCo/1*). The local IP address is 90.0.0.1; the IP address of the remote peer is 90.0.0.10. The baud rate is the default rate for the *tty* device. VJ compression and authentication have been disabled, and LCP echo requests have been enabled.

```
PPP_OPTIONS pppOpt;    /* PPP configuration options */
void routine ()
{
    pppOpt.flags = OPT_PASSIVE_MODE | OPT_NO_PAP | OPT_NO_CHAP | OPT_NO_VJ;
    pppOpt.lcp_echo_interval = "30";
    pppOpt.lcp_echo_failure = "10";
    pppInit (0, "/tyCo/1", "90.0.0.1", "90.0.0.10", 0, &pppOpt, NULL);
}
```

The following routine generates the same results as the previous example. The difference is that the configuration options are obtained from a file rather than a structure.

```
pppFile = "phobos:/tmp/ppp_options"; /* PPP configuration options file */
void routine ()
{
    pppInit (0, "/tyCo/1", "90.0.0.1", "90.0.0.10", 0, NULL, pppFile);
}
```

where **phobos:/tmp/ppp\_options** contains:

```
passive
no_pap
no_chap
no_vj
lcp_echo_interval 30
lcp_echo_failure 10
```

**RETURNS** OK, or **ERROR** if the PPP interface cannot be initialized because the daemon task cannot be spawned or memory is insufficient.

**SEE ALSO** **pppLib**, **pppShow**, **pppDelete()**, *VxWorks Programmer's Guide: Network*

---

## pppSecretAdd()

**NAME** **pppSecretAdd()** – add a secret to the PPP authentication secrets table

**SYNOPSIS**

```

STATUS pppSecretAdd
(
    char * client,           /* client being authenticated */
    char * server,         /* server performing authentication */
    char * secret,         /* secret used for authentication */
    char * addr,           /* acceptable client IP addresses */
)

```

**DESCRIPTION** This routine adds a secret to the Point-to-Point Protocol (PPP) authentication secrets table. This table may be used by the Password Authentication Protocol (PAP) and Challenge-Handshake Authentication Protocol (CHAP) user authentication protocols.

When a PPP link is established, a “server” may require a “client” to authenticate itself using a “secret”. Clients and servers obtain authentication secrets by searching secrets files, or by searching the secrets table constructed by this routine. Clients and servers search the secrets table by matching client and server names with table entries, and retrieving the associated secret.

Client and server names in the table consisting of “\*” are considered wildcards; they serve as matches for any client and/or server name if an exact match cannot be found.

If *secret* starts with “@”, *secret* is assumed to be the name of a file, wherein the actual secret can be read.

If *addr* is not **NULL**, it should contain a list of acceptable client IP addresses. When a server is authenticating a client and the client’s IP address is not contained in the list of acceptable addresses, the link is terminated. Any IP address will be considered acceptable if *addr* is **NULL**. If this parameter is “-”, all IP addresses are disallowed.

**RETURNS** OK, or **ERROR** if the secret cannot be added to the table.

**SEE ALSO** **pppSecretLib**, **pppSecretDelete()**, **pppSecretShow()**

## pppSecretDelete()

**NAME** `pppSecretDelete()` – delete a secret from the PPP authentication secrets table

**SYNOPSIS**

```
STATUS pppSecretDelete
(
    char * client,          /* client being authenticated */
    char * server,         /* server performing authentication */
    char * secret          /* secret used for authentication */
)
```

**DESCRIPTION** This routine deletes a secret from the Point-to-Point Protocol (PPP) authentication secrets table. When searching for a secret to delete from the table, the wildcard substitution (using “\*”) is not performed for client and/or server names. The *client*, *server*, and *secret* strings must match the table entry exactly in order to be deleted.

**RETURNS** OK, or ERROR if the table entry being deleted is not found.

**SEE ALSO** `pppSecretLib`, `pppSecretAdd()`, `pppSecretShow()`

---

## pppSecretShow()

**NAME** `pppSecretShow()` – display the PPP authentication secrets table

**SYNOPSIS**

```
void pppSecretShow (void)
```

**DESCRIPTION** This routine displays the Point-to-Point Protocol (PPP) authentication secrets table. The information in the secrets table may be used by the Password Authentication Protocol (PAP) and Challenge-Handshake Authentication Protocol (CHAP) user authentication protocols.

**RETURNS** N/A

**SEE ALSO** `pppShow`, `pppLib`, `pppSecretAdd()`, `pppSecretDelete()`

---

## pppstatGet()

**NAME** `pppstatGet()` – get PPP link statistics

**SYNOPSIS**

```
STATUS pppstatGet
(
    int          unit,          /* PPP interface unit number to examine */
    PPP_STAT * pStat          /* PPP_STAT structure to be filled */
)
```

**DESCRIPTION** This routine gets statistics for the specified Point-to-Point Protocol (PPP) link. Detailed are the numbers of bytes and packets received and sent through the PPP interface.

The PPP link statistics are returned through a `PPP_STAT` structure, which is defined in `h/netinet/ppp/pppShow.h`.

**RETURNS** `OK`, or `ERROR` if `unit` is an invalid PPP unit number.

**SEE ALSO** `pppShow`, `pppLib`

---

## pppstatShow()

**NAME** `pppstatShow()` – display PPP link statistics

**SYNOPSIS**

```
void pppstatShow (void)
```

**DESCRIPTION** This routine displays statistics for each initialized Point-to-Point Protocol (PPP) link. Detailed are the numbers of bytes and packets received and sent through each PPP interface.

**RETURNS** `N/A`

**SEE ALSO** `pppShow`, `pppLib`

## printErr()

<b>NAME</b>	<b>printErr()</b> – write a formatted string to the standard error stream
<b>SYNOPSIS</b>	<pre>int printErr (     const char * fmt,          /* format string to write */     ...                      /* optional arguments to format */ )</pre>
<b>DESCRIPTION</b>	This routine writes a formatted string to standard error. Its function and syntax are otherwise identical to <b>printf()</b> .
<b>RETURNS</b>	The number of characters output, or <b>ERROR</b> if there is an error during output.
<b>SEE ALSO</b>	<b>fioLib</b> , <b>printf()</b>

---

## printErrno()

<b>NAME</b>	<b>printErrno()</b> – print the definition of a specified error status value
<b>SYNOPSIS</b>	<pre>void printErrno (     int errno                 /* status code whose name is to be printed */ )</pre>
<b>DESCRIPTION</b>	<p>This command displays the error-status string, corresponding to a specified error-status value. It is only useful if the error-status symbol table has been built and included in the system. If <i>errno</i> is zero, then the current task status is used by calling <b>errnoGet()</b>.</p> <p>This facility is described in <b>errnoLib</b>.</p>
<b>RETURNS</b>	N/A
<b>SEE ALSO</b>	<b>usrLib</b> , <b>errnoLib</b> , <b>errnoGet()</b> , <i>VxWorks Programmer's Guide: Target Shell</i> , <b>windsh</b> , <i>Tornado User's Guide: Shell</i>

---

## printf()

**NAME** `printf()` – write a formatted string to the standard output stream (ANSI)

**SYNOPSIS**

```
int printf
(
    const char * fmt,          /* format string to write */
    ...                       /* optional arguments to format string */
)
```

**DESCRIPTION** This routine writes output to standard output under control of the string *fmt*. The string *fmt* contains ordinary characters, which are written unchanged, plus conversion specifications, which cause the arguments that follow *fmt* to be converted and printed as part of the formatted string.

The number of arguments for the format is arbitrary, but they must correspond to the conversion specifications in *fmt*. If there are insufficient arguments, the behavior is undefined. If the format is exhausted while arguments remain, the excess arguments are evaluated but otherwise ignored. The routine returns when the end of the format string is encountered.

The format is a multibyte character sequence, beginning and ending in its initial shift state. The format is composed of zero or more directives: ordinary multibyte characters (not %) that are copied unchanged to the output stream; and conversion specification, each of which results in fetching zero or more subsequent arguments. Each conversion specification is introduced by the % character. After the %, the following appear in sequence:

- Zero or more flags (in any order) that modify the meaning of the conversion specification.
- An optional minimum field width. If the converted value has fewer characters than the field width, it will be padded with spaces (by default) on the left (or right, if the left adjustment flag, described later, has been given) to the field width. The field width takes the form of an asterisk (\*) (described later) or a decimal integer.
- An optional precision that gives the minimum number of digits to appear for the **d**, **i**, **o**, **u**, **x**, and **X** conversions, the number of digits to appear after the decimal-point character for **e**, **E**, and **f** conversions, the maximum number of significant digits for the **g** and **G** conversions, or the maximum number of characters to be written from a string in the **s** conversion. The precision takes the form of a period (.) followed either by an asterisk (\*) (described later) or by an optional decimal integer; if only the period is specified, the precision is taken as zero. If a precision appears with any other conversion specifier, the behavior is undefined.
- An optional **h** specifying that a following **d**, **i**, **o**, **u**, **x**, and **X** conversion specifier applies to a **short int** or **unsigned short int** argument (the argument will have been

**printf()**

promoted according to the integral promotions, and its value converted to **short int** or **unsigned short int** before printing); an optional **h** specifying that a following **n** conversion specifier applies to a pointer to a **short int** argument. An optional **l** (ell) specifying that a following **d**, **i**, **o**, **u**, **x**, and **X** conversion specifier applies to a **long int** or **unsigned long int** argument; or an optional **l** specifying that a following **n** conversion specifier applies to a pointer to a **long int** argument. An optional **ll** (ell-ell) specifying that a following **d**, **i**, **o**, **u**, **x**, and **X** conversion specifier applies to a **long long int** or **unsigned long long int** argument; or an optional **ll** specifying that a following **n** conversion specifier applies to a pointer to a **long long int** argument. If a **h**, **l** or **ll** appears with any other conversion specifier, the behavior is undefined.

---

**WARNING:** ANSI C also specifies an optional **L** in some of the same contexts as **l** above, corresponding to a **long double** argument. However, the current release of VxWorks does not support **long double** data; using the optional **L** gives unpredictable results.

---

– A character that specifies the type of conversion to be applied.

As noted above, a field width, or precision, or both, can be indicated by an asterisk (\*). In this case, an **int** argument supplies the field width or precision. The arguments specifying field width or precision, or both, should appear (in that order) before the argument to be converted. A negative field width argument is taken as a - flag followed by a positive field width. A negative precision argument is taken as if the precision were omitted.

The flag characters and their meanings are:

- The result of the conversion will be left-justified within the field. (it will be right-justified if this flag is not specified.)
- + The result of a signed conversion will always begin with a plus or minus sign. (It will begin with a sign only when a negative value is converted if this flag is not specified.)

space

If the first character of a signed conversion is not a sign, or if a signed conversion results in no characters, a space will be prefixed to the result. If the **space** and **+** flags both appear, the **space** flag will be ignored.

#

The result is to be converted to an “alternate form.” For **o** conversion it increases the precision to force the first digit of the result to be a zero. For **x** (or **X**) conversion, a non-zero result will have “0x” (or “0X”) prefixed to it. For **e**, **E**, **f**, **g**, and **G** conversions, the result will always contain a decimal-point character, even if no digits follow it. (Normally, a decimal-point character appears in the result of these conversions only if no digit follows it). For **g** and **G** conversions, trailing zeros will not be removed from the result. For other conversions, the behavior is undefined.

0

For **d**, **i**, **o**, **u**, **x**, **X**, **e**, **E**, **f**, **g**, and **G** conversions, leading zeros (following any

indication of sign or base) are used to pad to the field width; no space padding is performed. If the **0** and **-** flags both appear, the **0** flag will be ignored. For **d**, **i**, **o**, **u**, **x**, and **X** conversions, if a precision is specified, the **0** flag will be ignored. For other conversions, the behavior is undefined.

The conversion specifiers and their meanings are:

**d, i**

The **int** argument is converted to signed decimal in the style **[-]dddd**. The precision specifies the minimum number of digits to appear; if the value being converted can be represented in fewer digits, it will be expanded with leading zeros. The default precision is 1. The result of converting a zero value with a precision of zero is no characters.

**o, u, x, X**

The **unsigned int** argument is converted to unsigned octal (**o**), unsigned decimal (**u**), or unsigned hexadecimal notation (**x** or **X**) in the style **dddd**; the letters **abcdef** are used for **x** conversion and the letters **ABCDEF** for **X** conversion. The precision specifies the minimum number of digits to appear; if the value being converted can be represented in fewer digits, it will be expanded with leading zeros. The default precision is 1. The result of converting a zero value with a precision of zero is no characters.

**f**

The **double** argument is converted to decimal notation in the style **[-]ddd.ddd**, where the number of digits after the decimal point character is equal to the precision specification. If the precision is missing, it is taken as 6; if the precision is zero and the **#** flag is not specified, no decimal-point character appears. If a decimal-point character appears, at least one digit appears before it. The value is rounded to the appropriate number of digits.

**e, E**

The **double** argument is converted in the style **[-]d.ddde+/-dd**, where there is one digit before the decimal-point character (which is non-zero if the argument is non-zero) and the number of digits after it is equal to the precision; if the precision is missing, it is taken as 6; if the precision is zero and the **#** flag is not specified, no decimal-point character appears. The value is rounded to the appropriate number of digits. The **E** conversion specifier will produce a number with **E** instead of **e** introducing the exponent. The exponent always contains at least two digits. If the value is zero, the exponent is zero.

**g, G**

The **double** argument is converted in style **f** or **e** (or in style **E** in the case of a **G** conversion specifier), with the precision specifying the number of significant digits. If the precision is zero, it is taken as 1. The style used depends on the value converted; style **e** (or **E**) will be used only if the exponent resulting from such a conversion is less than **-4** or greater than or equal to the precision. Trailing zeros are removed from the fractional portion of the result; a decimal-point character appears only if it is followed

**printf()**

by a digit.

**c**

The **int** argument is converted to **unsigned char**; the resulting character is written.

**s**

The argument should be a pointer to an array of character type. Characters from the array are written up to (but not including) a terminating null character; if the precision is specified, no more than that many characters are written. If the precision is not specified or is greater than the size of the array, the array will contain a null character.

**p**

The argument should be a pointer to **void**. The value of the pointer is converted to a sequence of printable characters, in hexadecimal representation (prefixed with "0x").

**n**

The argument should be a pointer to an integer into which the number of characters written to the output stream so far by this call to **fprintf()** is written. No argument is converted.

**%**

A % is written. No argument is converted. The complete conversion specification is %%.

If a conversion specification is invalid, the behavior is undefined.

If any argument is, or points to, a union or an aggregate (except for an array of character type using **s** conversion, or a pointer using **p** conversion), the behavior is undefined.

In no case does a non-existent or small field width cause truncation of a field if the result of a conversion is wider than the field width, the field is expanded to contain the conversion result.

**INCLUDE FILES**     **ioLib.h**

**RETURNS**             The number of characters written, or a negative value if an output error occurs.

**SEE ALSO**             **ioLib**, **fprintf()**, *American National Standard for Information Systems -Programming Language - C, ANSI X3.159-1989: Input/Output (stdio.h)*

---

## printLogo()

- NAME** `printLogo()` – print the VxWorks logo
- SYNOPSIS** `void printLogo (void)`
- DESCRIPTION** This command displays the VxWorks banner seen at boot time. It also displays the VxWorks version number and kernel version number.
- RETURNS** N/A
- SEE ALSO** `usrLib`, *VxWorks Programmer's Guide: Target Shell*, `windsh`, *Tornado User's Guide: Shell*

---

## proxyArpLibInit()

- NAME** `proxyArpLibInit()` – initialize proxy ARP
- SYNOPSIS** `STATUS proxyArpLibInit`
- ```
(  
    int clientSizeLog2,      /* client table size as power of two */  
    int portSizeLog2        /* port table size as power of two */  
)
```
- DESCRIPTION** This routine starts the proxy ARP server by initializing the required data structures and installing the necessary input hooks. It should be called only once; subsequent calls have no effect. The `clientSizeLog2` and `portSizeLog2` parameters specify the internal hash table sizes. Each must be equal to a power of two, or zero to use a default size value.
- RETURNS** OK, or ERROR if unsuccessful.
- SEE ALSO** `proxyArpLib`

## proxyNetCreate()

**NAME** proxyNetCreate() – create a proxy ARP network

**SYNOPSIS**

```
STATUS proxyNetCreate
(
    char * proxyAddr,      /* address of proxy network interface */
    char * mainAddr       /* address of main network interface */
)
```

**DESCRIPTION** This routine activates proxy services between the proxy network connected to the interface with the *proxyAddr* IP address and the main network connected to the interface with the *mainAddr* address. Once registration is complete, the proxy server will disguise the physically separated networks as a single logical network.

The corresponding interfaces must be attached and configured with IP addresses before calling this routine. If the proxy network shares the same logical subnet number as the main network, the corresponding interface to the proxy network must use a value of 255.255.255.255 for the netmask.

**RETURNS** OK, or ERROR if unsuccessful.

**ERRNO** S\_proxyArpLib\_INVALID\_ADDRESS

**SEE ALSO** proxyArpLib

---

## proxyNetDelete()

**NAME** proxyNetDelete() – delete a proxy network

**SYNOPSIS**

```
STATUS proxyNetDelete
(
    char * proxyAddr      /* proxy net address */
)
```

**DESCRIPTION** This routine deletes the proxy network specified by *proxyAddr*. It also removes all the proxy clients that exist on that network.

**RETURNS** OK, or ERROR if unsuccessful.

**SEE ALSO** proxyArpLib

---

## proxyNetShow()

**NAME** `proxyNetShow()` – show proxy ARP networks

**SYNOPSIS** `void proxyNetShow (void)`

**DESCRIPTION** This routine displays the proxy networks and their associated clients.

**EXAMPLE**

```
-> proxyNetShow
main interface 147.11.1.182 proxy interface 147.11.1.183
client 147.11.1.184
```

**RETURNS** N/A

**SEE ALSO** `proxyArpLib`

---

## proxyPortFwdOff()

**NAME** `proxyPortFwdOff()` – disable broadcast forwarding for a particular port

**SYNOPSIS**

```
STATUS proxyPortFwdOff
(
    int port          /* port number */
)
```

**DESCRIPTION** This routine disables broadcast forwarding on port number *port*. To disable the (previously enabled) forwarding of all ports via `proxyPortFwdOn()`, specify zero for *port*.

**RETURNS** OK, or ERROR if unsuccessful.

**SEE ALSO** `proxyArpLib`

## proxyPortFwdOn()

**NAME** proxyPortFwdOn() – enable broadcast forwarding for a particular port

**SYNOPSIS**

```
STATUS proxyPortFwdOn
(
    int port                /* port number */
)
```

**DESCRIPTION** This routine enables broadcasts destined for the port, *port*, to be forwarded to and from the proxy network. To enable all ports, specify zero for *port*.

**RETURNS** OK, or ERROR if unsuccessful.

**SEE ALSO** proxyArpLib

---

## proxyPortShow()

**NAME** proxyPortShow() – show ports enabled for broadcast forwarding

**SYNOPSIS** void proxyPortShow (void)

**DESCRIPTION** This routine displays the destination ports for which the proxy ARP server will forward broadcast messages between the physically separate networks.

**EXAMPLE**

```
-> proxyPortShow
enabled ports:
port 67
```

**RETURNS** N/A

**SEE ALSO** proxyArpLib

---

## proxyReg()

**NAME** proxyReg() – register a proxy client

**SYNOPSIS**

```
STATUS proxyReg
(
    char * ifName,          /* interface name */
    char * proxyAddr       /* proxy address */
)
```

**DESCRIPTION** This routine sends a message over the network interface *ifName* to register *proxyAddr* as a proxy client.

**RETURNS** OK, or ERROR if unsuccessful.

**SEE ALSO** proxyLib

---

## proxyUnreg()

**NAME** proxyUnreg() – unregister a proxy client

**SYNOPSIS**

```
STATUS proxyUnreg
(
    char * ifName,          /* interface name */
    char * proxyAddr       /* proxy address */
)
```

**DESCRIPTION** This routine sends a message over the network interface *ifName* to unregister *proxyAddr* as a proxy client.

**RETURNS** OK, or ERROR if unsuccessful.

**SEE ALSO** proxyLib

---

## psrShow()

**NAME** `psrShow()` – display the meaning of a specified `psr` value, symbolically (ARM)

**SYNOPSIS**

```
void psrShow
(
    ULONG psrValue          /* psr value to show */
)
```

**DESCRIPTION** This routine displays the meaning of all the fields in a specified `psr` value, symbolically.  
Extracted from `psl.h`:

Definition of bits in the Sun-4 PSR (Processor Status Register)

```
-----
| IMPL | VER |           ICC           | resvd | EC | EF | PIL | S | PS | ET | CWP |
|-----|-----| N | Z | V | C |-----|-----|-----|-----|-----|-----|
|-----|-----|---|---|---|---|-----|-----|-----|-----|-----|
31  28 27 24  23  22  21  20 19  14  13  12  11  8  7  6  5  4  0
```

For compatibility with future revisions, reserved bits are defined to be initialized to zero and, if written, must be preserved.

**EXAMPLE**

```
-> psrShow 0x00001FE7
Implementation 0, mask version 0:
Fujitsu MB86900 or LSI L64801, 7 windows
    no SWAP, FSQRT, CP, extended fp instructions
    Condition codes: . . . .
    Coprocessor enables: . EF
    Processor interrupt level: f
    Flags: S PS ET
    Current window pointer: 0x07
->
```

**RETURNS** N/A

**SEE ALSO** `dbgArchLib`, `psr()`, *ARM Architecture Reference Manual*

---

## pthreadLibInit()

|                    |                                                                                                                                                                                                                                 |
|--------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>        | <b>pthreadLibInit()</b> – initialize POSIX threads support                                                                                                                                                                      |
| <b>SYNOPSIS</b>    | <code>void pthreadLibInit (void)</code>                                                                                                                                                                                         |
| <b>DESCRIPTION</b> | This routine initializes the POSIX threads ( <i>threads</i> ) support for VxWorks. It should be called before any POSIX threads functions are used; normally it will be called as part of the kernel's initialization sequence. |
| <b>RETURNS</b>     | N/A                                                                                                                                                                                                                             |
| <b>SEE ALSO</b>    | <b>pthreadLib</b>                                                                                                                                                                                                               |

---

## pthread\_attr\_destroy()

|                    |                                                                                                                |
|--------------------|----------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>        | <b>pthread_attr_destroy()</b> – destroy a thread attributes object (POSIX)                                     |
| <b>SYNOPSIS</b>    | <pre>int pthread_attr_destroy (     pthread_attr_t * pAttr    /* thread attributes */ )</pre>                  |
| <b>DESCRIPTION</b> | Destroy the thread attributes object <i>pAttr</i> . It should not be re-used until it has been re-initialized. |
| <b>RETURNS</b>     | On success zero; on failure a non-zero error code.                                                             |
| <b>ERRNOS</b>      | EINVAL                                                                                                         |
| <b>SEE ALSO</b>    | <b>pthreadLib, pthread_attr_init()</b>                                                                         |

---

## pthread\_attr\_getdetachstate()

**NAME** pthread\_attr\_getdetachstate() – get value of detachstate attribute from thread attributes object (POSIX)

**SYNOPSIS**

```
int pthread_attr_getdetachstate
(
    const pthread_attr_t * pAttr,          /* thread attributes */
    int *                  pDetachstate /* current detach state (out) */
)
```

**DESCRIPTION** This routine returns the current detach state specified in the thread attributes object *pAttr*. The value is stored in the location pointed to by *pDetachstate*. Possible values for the detach state are: `PTHREAD_CREATE_DETACHED` and `PTHREAD_CREATE_JOINABLE`.

**RETURNS** Always returns zero.

**ERRNOS** None.

**SEE ALSO** pthreadLib, pthread\_attr\_init(), pthread\_attr\_setdetachstate()

---

## pthread\_attr\_getinheritsched()

**NAME** pthread\_attr\_getinheritsched() – get value of inheritsched attribute in thread attributes object (POSIX)

**SYNOPSIS**

```
int pthread_attr_getinheritsched
(
    const pthread_attr_t * pAttr,          /* thread attributes object */
    int *                  pInheritsched /* inheritance mode (out) */
)
```

**DESCRIPTION** This routine gets the scheduling inheritance value from the thread attributes object *pAttr*. Possible values are:

**PTHREAD\_INHERIT\_SCHED**

Inherit scheduling parameters from parent thread.

**PTHREAD\_EXPLICIT\_SCHED**

Use explicitly provided scheduling parameters (*i.e.*, those specified in the thread attributes object).

**RETURNS** On success zero; on failure a non-zero error code.

**ERRNOS** EINVAL

**SEE ALSO** pthreadLib, pthread\_attr\_init(), pthread\_attr\_getschedparam(), pthread\_attr\_getschedpolicy(), pthread\_attr\_setinheritsched()

---

## pthread\_attr\_getname()

**NAME** pthread\_attr\_getname() – get name of thread attribute object

**SYNOPSIS**

```
int pthread_attr_getname
(
    pthread_attr_t * pAttr,
    char *          *name
)
```

**DESCRIPTION** This routine gets the name in the specified thread attributes object, *pAttr*.

**RETURNS** Always returns zero

**ERRNOS** None.

**SEE ALSO** pthreadLib, pthread\_attr\_setname(),

---

## pthread\_attr\_getschedparam()

**NAME** pthread\_attr\_getschedparam() – get value of schedparam attribute from thread attributes object (POSIX)

**SYNOPSIS**

```
int pthread_attr_getschedparam
(
    const pthread_attr_t * pAttr, /* thread attributes */
    struct sched_param *  pParam /* current parameters (out) */
)
```

**DESCRIPTION** Return, via the pointer *pParam*, the current scheduling parameters from the thread attributes object *pAttr*.

**RETURNS** On success zero; on failure a non-zero error code.

**ERRNOS** EINVAL

**SEE ALSO** pthreadLib, schedPxLib, pthread\_attr\_init(), pthread\_attr\_setschedparam(), pthread\_getschedparam(), pthread\_setschedparam(), sched\_getparam(), sched\_setparam()

---

## pthread\_attr\_getschedpolicy()

**NAME** pthread\_attr\_getschedpolicy() – get schedpolicy attribute from thread attributes object (POSIX)

**SYNOPSIS**

```
int pthread_attr_getschedpolicy
(
    const pthread_attr_t * pAttr, /* thread attributes */
    int * pPolicy /* current policy (out) */
)
```

**DESCRIPTION** This routine returns, via the pointer *pPolicy*, the current scheduling policy in the thread attributes object specified by *pAttr*. Possible values for VxWorks systems are SCHED\_RR and SCHED\_FIFO; SCHED\_OTHER is not supported.

**RETURNS** On success zero; on failure a non-zero error code.

**ERRNOS** EINVAL

**SEE ALSO** pthreadLib, schedPxLib, pthread\_attr\_init(), pthread\_attr\_setschedpolicy(), pthread\_getschedparam(), pthread\_setschedparam(), sched\_setscheduler(), sched\_getscheduler()

---

## pthread\_attr\_getscope()

- NAME** pthread\_attr\_getscope() – get contention scope from thread attributes (POSIX)
- SYNOPSIS**
- ```
int pthread_attr_getscope
(
    const pthread_attr_t * pAttr,          /* thread attributes object */
    int *                  pContentionScope /* contention scope (out) */
)
```
- DESCRIPTION** Reads the current contention scope setting from a thread attributes object. For VxWorks this is always `PTHREAD_SCOPE_SYSTEM`. If the thread attributes object is uninitialized then `EINVAL` will be returned. The contention scope is returned in the location pointed to by *pContentionScope*.
- RETURNS** On success zero; on failure a non-zero error code.
- ERRNOS** `EINVAL`
- SEE ALSO** `pthreadLib`, `pthread_attr_init()`, `pthread_attr_setscope()`

---

## pthread\_attr\_getstackaddr()

- NAME** pthread\_attr\_getstackaddr() – get value of stackaddr attribute from thread attributes object (POSIX)
- SYNOPSIS**
- ```
int pthread_attr_getstackaddr
(
    const pthread_attr_t * pAttr,          /* thread attributes */
    void *                  *ppStackaddr /* current stack address (out) */
)
```
- DESCRIPTION** This routine returns the stack address from the thread attributes object *pAttr* in the location pointed to by *ppStackaddr*.
- RETURNS** Always returns zero.
- ERRNOS** None.
- SEE ALSO** `pthreadLib`, `pthread_attr_init()`, `pthread_attr_setstacksize()`

---

## **pthread\_attr\_getstacksize()**

|                    |                                                                                                                                                                                 |
|--------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>        | <b>pthread_attr_getstacksize()</b> – get stack value of stacksize attribute from thread attributes object (POSIX)                                                               |
| <b>SYNOPSIS</b>    | <pre>int pthread_attr_getstacksize (     const pthread_attr_t * pAttr,    /* thread attributes */     size_t *                pStacksize /* current stack size (out) */ )</pre> |
| <b>DESCRIPTION</b> | This routine gets the current stack size from the thread attributes object <i>pAttr</i> and places it in the location pointed to by <i>pStacksize</i> .                         |
| <b>RETURNS</b>     | Always returns zero.                                                                                                                                                            |
| <b>ERRNOS</b>      | None.                                                                                                                                                                           |
| <b>SEE ALSO</b>    | <b>pthreadLib</b> , <b>pthread_attr_init()</b> , <b>pthread_attr_setstacksize()</b>                                                                                             |

---

## **pthread\_attr\_init()**

|                    |                                                                                                                                                                                                                                                                                                                                                                                                                                              |
|--------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>        | <b>pthread_attr_init()</b> – initialize thread attributes object (POSIX)                                                                                                                                                                                                                                                                                                                                                                     |
| <b>SYNOPSIS</b>    | <pre>int pthread_attr_init (     pthread_attr_t * pAttr    /* thread attributes */ )</pre>                                                                                                                                                                                                                                                                                                                                                   |
| <b>DESCRIPTION</b> | <p>This routine initializes a thread attributes object. If <i>pAttr</i> is <b>NULL</b> then this function will return <b>EINVAL</b>.</p> <p>The attributes that are set by default are as follows:</p> <p>Stack Address<br/>    <b>NULL</b> - allow the system to allocate the stack.</p> <p>Stack Size<br/>    <b>0</b> - use the VxWorks <b>taskLib</b> default stack size.</p> <p>Detach State<br/>    <b>PTHREAD_CREATE_JOINABLE</b></p> |

Contention Scope

`PTHREAD_SCOPE_SYSTEM`

Scheduling Inheritance

`PTHREAD_INHERIT_SCHED`

Scheduling Policy

`SCHED_RR`

Scheduling Priority

Use `pthreadLib` default priority

Note that the scheduling policy and priority values are only used if the scheduling inheritance mode is changed to `PTHREAD_EXPLICIT_SCHED` - see `pthread_attr_setinheritsched()` for information.

**RETURNS** On success zero; on failure a non-zero error code.

**ERRNOS** `EINVAL`

**SEE ALSO** `pthreadLib`, `pthread_attr_destroy()`, `pthread_attr_getdetachstate()`, `pthread_attr_getinheritsched()`, `pthread_attr_getschedparam()`, `pthread_attr_getschedpolicy()`, `pthread_attr_getscope()`, `pthread_attr_getstackaddr()`, `pthread_attr_getstacksize()`, `pthread_attr_setdetachstate()`, `pthread_attr_setinheritsched()`, `pthread_attr_setschedparam()`, `pthread_attr_setschedpolicy()`, `pthread_attr_setscope()`, `pthread_attr_setstackaddr()`, `pthread_attr_setstacksize()`

P

---

## `pthread_attr_setdetachstate()`

**NAME** `pthread_attr_setdetachstate()` – set detachstate attribute in thread attributes object (POSIX)

**SYNOPSIS**

```
int pthread_attr_setdetachstate
(
    pthread_attr_t * pAttr,      /* thread attributes */
    int             detachstate /* new detach state */
)
```

**DESCRIPTION** This routine sets the detach state in the thread attributes object *pAttr*. The new detach state specified by *detachstate* must be one of `PTHREAD_CREATE_DETACHED` or `PTHREAD_CREATE_JOINABLE`. Any other values will cause an error to be returned (`EINVAL`).

**RETURNS** On success zero; on failure a non-zero error code.

**ERRNOS**           EINVAL

**SEE ALSO**        pthreadLib, pthread\_attr\_getdetachstate(), pthread\_attr\_init()

---

## pthread\_attr\_setinheritsched()

**NAME**            pthread\_attr\_setinheritsched() – set inheritsched attribute in thread attribute object  
                  (POSIX)

**SYNOPSIS**

```
int pthread_attr_setinheritsched
(
    pthread_attr_t * pAttr,      /* thread attributes object */
    int             inheritsched /* inheritance mode */
)
```

**DESCRIPTION**    This routine sets the scheduling inheritance to be used when creating a thread with the thread attributes object specified by *pAttr*.

Possible values are:

**PTHREAD\_INHERIT\_SCHED**

Inherit scheduling parameters from parent thread.

**PTHREAD\_EXPLICIT\_SCHED**

Use explicitly provided scheduling parameters (*i.e.*, those specified in the thread attributes object).

**RETURNS**        On success zero; on failure a non-zero error code.

**ERRNOS**           EINVAL

**SEE ALSO**        pthreadLib, pthread\_attr\_getinheritsched(), pthread\_attr\_init(),  
pthread\_attr\_setschedparam(), pthread\_attr\_setschedpolicy()

---

## pthread\_attr\_setname()

**NAME** pthread\_attr\_setname() – set name in thread attribute object

**SYNOPSIS**

```
int pthread_attr_setname
(
    pthread_attr_t * pAttr,
    char *          name
)
```

**DESCRIPTION** This routine sets the name in the specified thread attributes object, *pAttr*.

**RETURNS** Always returns zero.

**ERRNOS** None.

**SEE ALSO** pthreadLib, pthread\_attr\_getname()

---

## pthread\_attr\_setschedparam()

**NAME** pthread\_attr\_setschedparam() – set schedparam attribute in thread attributes object (POSIX)

**SYNOPSIS**

```
int pthread_attr_setschedparam
(
    pthread_attr_t * pAttr, /* thread attributes */
    const struct sched_param * pParam /* new parameters */
)
```

**DESCRIPTION** Set the scheduling parameters in the thread attributes object *pAttr*. The scheduling parameters are essentially the thread's priority.

**RETURNS** On success zero; on failure a non-zero error code.

**ERRNOS** EINVAL

**SEE ALSO** pthreadLib, schedPxLib, pthread\_attr\_getschedparam(), pthread\_attr\_init(), pthread\_getschedparam(), pthread\_setschedparam(), sched\_getparam(), sched\_setparam()

---

## pthread\_attr\_setschedpolicy()

**NAME** pthread\_attr\_setschedpolicy() – set schedpolicy attribute in thread attributes object (POSIX)

**SYNOPSIS**

```
int pthread_attr_setschedpolicy
(
    pthread_attr_t * pAttr, /* thread attributes */
    int            policy /* new policy */
)
```

**DESCRIPTION** Select the thread scheduling policy. The default scheduling policy is to inherit the current system setting. Unlike the POSIX model, scheduling policies under VxWorks are global. If a scheduling policy is being set explicitly, the `PTHREAD_EXPLICIT_SCHED` mode must be set (see `pthread_attr_setinheritsched()` for information), and the selected scheduling policy must match the global scheduling policy in place at the time; failure to do so will result in `pthread_create()` failing with the non-POSIX error `ENOTTY`.

POSIX defines the following policies:

**SCHED\_RR**

Real-time, round-robin scheduling.

**SCHED\_FIFO**

Real-time, first-in first-out scheduling.

**SCHED\_OTHER**

Other, non-real-time scheduling.

VxWorks only supports `SCHED_RR` and `SCHED_FIFO`.

**RETURNS** On success zero; on failure a non-zero error code.

**ERRNOS** EINVAL

**SEE ALSO** pthreadLib, schedPxLib, pthread\_attr\_getschedpolicy(), pthread\_attr\_init(), pthread\_attr\_setinheritsched(), pthread\_getschedparam(), pthread\_setschedparam(), sched\_setscheduler(), sched\_getscheduler()

---

## pthread\_attr\_setscope()

- NAME** `pthread_attr_setscope()` – set contention scope for thread attributes (POSIX)
- SYNOPSIS**
- ```
int pthread_attr_setscope
(
    pthread_attr_t * pAttr,          /* thread attributes object */
    int             contentionScope /* new contention scope */
)
```
- DESCRIPTION** For VxWorks PTHREAD\_SCOPE\_SYSTEM is the only supported contention scope. Any other value passed to this function will result in EINVAL being returned.
- RETURNS** On success zero; on failure a non-zero error code.
- ERRNOS** EINVAL
- SEE ALSO** `pthreadLib`, `pthread_attr_getscope()`, `pthread_attr_init()`

---

## pthread\_attr\_setstackaddr()

- NAME** `pthread_attr_setstackaddr()` – set stackaddr attribute in thread attributes object (POSIX)
- SYNOPSIS**
- ```
int pthread_attr_setstackaddr
(
    pthread_attr_t * pAttr,          /* thread attributes */
    void *          pStackaddr /* new stack address */
)
```
- DESCRIPTION** This routine sets the stack address in the thread attributes object *pAttr* to be *pStackaddr*. Note that the size of this stack must be large enough to also include the task's TCB. The size of the TCB varies by architecture but can be determined by calling `sizeof (WIND_TCB)`. Set stack size using the routine `pthread_attr_setstacksize()`.
- RETURNS** Zero, always.
- ERRNOS** None.
- SEE ALSO** `pthreadLib`, `pthread_attr_getstacksize()`, `pthread_attr_setstacksize()`, `pthread_attr_init()`

---

## pthread\_attr\_setstacksize()

**NAME** pthread\_attr\_setstacksize() – set stacksize attribute in thread attributes object (POSIX)

**SYNOPSIS**

```
int pthread_attr_setstacksize
(
    pthread_attr_t * pAttr,    /* thread attributes */
    size_t          stacksize /* new stack size */
)
```

**DESCRIPTION** This routine sets the thread stack size in the specified thread attributes object, *pAttr*.

**RETURNS** Always returns zero.

**ERRNOS** None.

**SEE ALSO** pthreadLib, pthread\_attr\_getstacksize(), pthread\_attr\_init()

---

## pthread\_cancel()

**NAME** pthread\_cancel() – cancel execution of a thread (POSIX)

**SYNOPSIS**

```
int pthread_cancel
(
    pthread_t thread    /* thread to cancel */
)
```

**DESCRIPTION** This routine sends a cancellation request to the thread specified by *thread*. Depending on the settings of that thread, it may ignore the request, terminate immediately or defer termination until it reaches a cancellation point. When the thread terminates it performs as if *pthread\_exit()* had been called with the exit status *PTHREAD\_CANCELED*.

**NOTE:** In VxWorks, asynchronous thread cancellation is accomplished using a signal. The signal *SIGCNCL* has been reserved for this purpose. Applications should take care not to block or handle this signal.

---

**RETURNS** On success zero; on failure a non-zero error code.

**ERRNOS** ESRCH

**SEE ALSO** pthreadLib, pthread\_exit(), pthread\_setcancelstate(), pthread\_setcanceltype(), pthread\_testcancel()

---

## pthread\_cleanup\_pop()

**NAME** pthread\_cleanup\_pop() – pop a cleanup routine off the top of the stack (POSIX)

**SYNOPSIS**

```
void pthread_cleanup_pop
(
    int run                /* execute handler? */
)
```

**DESCRIPTION** This routine removes the cleanup handler routine at the top of the cancellation cleanup stack of the calling thread and executes it if *run* is non-zero. The routine should have been added using the `pthread_cleanup_push()` function.

Once the routine is removed from the stack it is no longer called when the thread exits.

**RETURNS** N/A

**ERRNOS** N/A

**SEE ALSO** pthreadLib, pthread\_cleanup\_push(), pthread\_exit()

---

## pthread\_cleanup\_push()

**NAME** pthread\_cleanup\_push() – pushes a routine onto the cleanup stack (POSIX)

**SYNOPSIS**

```
void pthread_cleanup_push
(
    void (* routine)(void * ), /* cleanup routine */
    void * arg                /* argument */
)
```

**DESCRIPTION** This routine pushes the specified cancellation cleanup handler routine, *routine*, onto the cancellation cleanup stack of the calling thread. When a thread exits and its cancellation cleanup stack is not empty, the cleanup handlers are invoked with the argument *arg* in LIFO order from the cancellation cleanup stack.

**RETURNS** N/A

**ERRNOS** N/A

**SEE ALSO** pthreadLib, pthread\_cleanup\_pop(), pthread\_exit()

## pthread\_cond\_broadcast()

|                    |                                                                                                                                                                                                                                                         |
|--------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>        | <b>pthread_cond_broadcast()</b> – unblock all threads waiting on a condition (POSIX)                                                                                                                                                                    |
| <b>SYNOPSIS</b>    | <pre>int pthread_cond_broadcast (     pthread_cond_t * pCond )</pre>                                                                                                                                                                                    |
| <b>DESCRIPTION</b> | This function unblocks all threads blocked on the condition variable <i>pCond</i> . Nothing happens if no threads are waiting on the specified condition variable.                                                                                      |
| <b>RETURNS</b>     | On success zero; on failure a non-zero error code.                                                                                                                                                                                                      |
| <b>ERRNOS</b>      | EINVAL                                                                                                                                                                                                                                                  |
| <b>SEE ALSO</b>    | <b>pthreadLib</b> , <b>pthread_condattr_init()</b> , <b>pthread_condattr_destroy()</b> ,<br><b>pthread_cond_destroy()</b> , <b>pthread_cond_init()</b> , <b>pthread_cond_signal()</b> ,<br><b>pthread_cond_timedwait()</b> , <b>pthread_cond_wait()</b> |

---

## pthread\_cond\_destroy()

|                    |                                                                                                                                                                                                                                                                               |
|--------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>        | <b>pthread_cond_destroy()</b> – destroy a condition variable (POSIX)                                                                                                                                                                                                          |
| <b>SYNOPSIS</b>    | <pre>int pthread_cond_destroy (     pthread_cond_t * pCond    /* condition variable */ )</pre>                                                                                                                                                                                |
| <b>DESCRIPTION</b> | This routine destroys the condition variable pointed to by <i>pCond</i> . No threads can be waiting on the condition variable when this function is called. If there are threads waiting on the condition variable, then <b>pthread_cond_destroy()</b> returns <b>EBUSY</b> . |
| <b>RETURNS</b>     | On success zero; on failure a non-zero error code.                                                                                                                                                                                                                            |
| <b>ERRNOS</b>      | EINVAL, EBUSY                                                                                                                                                                                                                                                                 |
| <b>SEE ALSO</b>    | <b>pthreadLib</b> , <b>pthread_condattr_init()</b> , <b>pthread_condattr_destroy()</b> ,<br><b>pthread_cond_broadcast()</b> , <b>pthread_cond_init()</b> , <b>pthread_cond_signal()</b> ,<br><b>pthread_cond_timedwait()</b> , <b>pthread_cond_wait()</b>                     |

---

## pthread\_cond\_init()

**NAME** pthread\_cond\_init() – initialize condition variable (POSIX)

**SYNOPSIS**

```
int pthread_cond_init
(
    pthread_cond_t *    pCond, /* condition variable */
    pthread_condattr_t * pAttr /* condition variable attributes */
)
```

**DESCRIPTION**

This function initializes a condition variable. A condition variable is a synchronization device that allows threads to block until some predicate on shared data is satisfied. The basic operations on conditions are to signal the condition (when the predicate becomes true), and wait for the condition, blocking the thread until another thread signals the condition.

A condition variable must always be associated with a mutex to avoid a race condition between the wait and signal operations.

If *pAttr* is NULL then the default attributes are used as specified by POSIX; if *pAttr* is non-NULL then it is assumed to point to a condition attributes object initialized by **pthread\_condattr\_init()**, and those are the attributes used to create the condition variable.

**RETURNS** On success zero; on failure a non-zero error code.

**ERRNOS** EINVAL, EBUSY

**SEE ALSO** pthreadLib, pthread\_condattr\_init(), pthread\_condattr\_destroy(), pthread\_cond\_broadcast(), pthread\_cond\_destroy(), pthread\_cond\_signal(), pthread\_cond\_timedwait(), pthread\_cond\_wait()

## pthread\_cond\_signal()

**NAME** pthread\_cond\_signal() – unblock a thread waiting on a condition (POSIX)

**SYNOPSIS**

```
int pthread_cond_signal
(
    pthread_cond_t * pCond
)
```

**DESCRIPTION** This routine unblocks one thread waiting on the specified condition variable *pCond*. If no threads are waiting on the condition variable then this routine does nothing; if more than one thread is waiting, then one will be released, but it is not specified which one.

**RETURNS** On success zero; on failure a non-zero error code.

**ERRNOS** EINVAL

**SEE ALSO** pthreadLib, pthread\_condattr\_init(), pthread\_condattr\_destroy(), pthread\_cond\_broadcast(), pthread\_cond\_destroy(), pthread\_cond\_init(), pthread\_cond\_timedwait(), pthread\_cond\_wait()

---

## pthread\_cond\_timedwait()

**NAME** pthread\_cond\_timedwait() – wait for a condition variable with a timeout (POSIX)

**SYNOPSIS**

```
int pthread_cond_timedwait
(
    pthread_cond_t *      pCond, /* condition variable */
    pthread_mutex_t *     pMutex, /* POSIX mutex */
    const struct timespec * pAbstime /* timeout time */
)
```

**DESCRIPTION** This function atomically releases the mutex *pMutex* and waits for another thread to signal the condition variable *pCond*. As with **pthread\_cond\_wait()**, the mutex must be locked by the calling thread when **pthread\_cond\_timedwait()** is called.

If the condition variable is signalled before the system time reaches the time specified by *pAbsTime*, then the mutex is re-acquired and the calling thread unblocked.

If the system time reaches or exceeds the time specified by *pAbsTime* before the condition is signalled, then the mutex is re-acquired, the thread unblocked and **ETIMEDOUT** returned.

---

**NOTE:** The timeout is specified as an absolute value of the system clock in a *timespec* structure (see **clock\_gettime()** for more information). This is different from most VxWorks timeouts which are specified in ticks relative to the current time.

---

**RETURNS** On success zero; on failure a non-zero error code.

**ERRNOS** EINVAL, ETIMEDOUT

**SEE ALSO** pthreadLib, pthread\_condattr\_init(), pthread\_condattr\_destroy(), pthread\_cond\_broadcast(), pthread\_cond\_destroy(), pthread\_cond\_init(), pthread\_cond\_signal(), pthread\_cond\_wait()

---

## pthread\_cond\_wait()

**NAME** pthread\_cond\_wait() – wait for a condition variable (POSIX)

**SYNOPSIS**

```
int pthread_cond_wait
(
    pthread_cond_t * pCond, /* condition variable */
    pthread_mutex_t * pMutex /* POSIX mutex */
)
```

**DESCRIPTION** This function atomically releases the mutex *pMutex* and waits for the condition variable *pCond* to be signalled by another thread. The mutex must be locked by the calling thread when **pthread\_cond\_wait()** is called; if it is not then this function returns an error (EINVAL).

Before returning to the calling thread, **pthread\_cond\_wait()** re-acquires the mutex.

**RETURNS** On success zero; on failure a non-zero error code.

**ERRNOS** EINVAL

**SEE ALSO** pthreadLib, pthread\_condattr\_init(), pthread\_condattr\_destroy(), pthread\_cond\_broadcast(), pthread\_cond\_destroy(), pthread\_cond\_init(), pthread\_cond\_signal(), pthread\_cond\_timedwait()

## **pthread\_condattr\_destroy()**

**NAME** `pthread_condattr_destroy()` – destroy a condition attributes object (POSIX)

**SYNOPSIS**

```
int pthread_condattr_destroy
(
    pthread_condattr_t * pAttr /* condition variable attributes */
)
```

**DESCRIPTION** This routine destroys the condition attribute object *pAttr*. It must not be reused until it is re-initialized.

**RETURNS** Always returns zero.

**ERRNOS** None.

**SEE ALSO** `pthreadLib`, `pthread_cond_init()`, `pthread_condattr_init()`

---

## **pthread\_condattr\_init()**

**NAME** `pthread_condattr_init()` – initialize a condition attribute object (POSIX)

**SYNOPSIS**

```
int pthread_condattr_init
(
    pthread_condattr_t * pAttr /* condition variable attributes */
)
```

**DESCRIPTION** This routine initializes the condition attribute object *pAttr* and fills it with default values for the attributes.

**RETURNS** On success zero; on failure a non-zero error code.

**ERRNOS** EINVAL

**SEE ALSO** `pthreadLib`, `pthread_cond_init()`, `pthread_condattr_destroy()`

---

## pthread\_create()

**NAME** pthread\_create() – create a thread (POSIX)

**SYNOPSIS**

```
int pthread_create
(
    pthread_t *      pThread,    /* Thread ID (out) */
    const pthread_attr_t * pAttr, /* Thread attributes object */
    void * (* startRoutine)(void * ), /* Entry function */
    void *          arg         /* Entry function argument */
)
```

**DESCRIPTION** This routine creates a new thread and if successful writes its ID into the location pointed to by *pThread*. If *pAttr* is NULL then default attributes are used. The new thread executes *startRoutine* with *arg* as its argument.

**RETURNS** On success zero; on failure a non-zero error code.

**ERRNOS** EINVAL, EAGAIN

**SEE ALSO** pthreadLib, pthread\_exit(), pthread\_join()

---

## pthread\_detach()

**NAME** pthread\_detach() – dynamically detach a thread (POSIX)

**SYNOPSIS**

```
int pthread_detach
(
    pthread_t thread    /* thread to detach */
)
```

**DESCRIPTION** This routine puts the thread *thread* into the detached state. This prevents other threads from synchronizing on the termination of the thread using **pthread\_join()**.

**RETURNS** On success zero; on failure a non-zero error code.

**ERRNOS** EINVAL, ESRCH

**SEE ALSO** pthreadLib, pthread\_join()

## pthread\_equal()

|                    |                                                                                                                         |
|--------------------|-------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>        | <b>pthread_equal()</b> – compare thread IDs (POSIX)                                                                     |
| <b>SYNOPSIS</b>    | <pre>int pthread_equal (     pthread_t t1,          /* thread one */     pthread_t t2          /* thread two */ )</pre> |
| <b>DESCRIPTION</b> | Tests the equality of the two threads <i>t1</i> and <i>t2</i> .                                                         |
| <b>RETURNS</b>     | Non-zero if <i>t1</i> and <i>t2</i> refer to the same thread, otherwise zero.                                           |
| <b>SEE ALSO</b>    | <b>pthreadLib</b>                                                                                                       |

---

## pthread\_exit()

|                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
|--------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>        | <b>pthread_exit()</b> – terminate a thread (POSIX)                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
| <b>SYNOPSIS</b>    | <pre>void pthread_exit (     void * status          /* exit status */ )</pre>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
| <b>DESCRIPTION</b> | <p>This function terminates the calling thread. All cleanup handlers that have been set for the calling thread with <b>pthread_cleanup_push()</b> are executed in reverse order (the most recently added handler is executed first). Termination functions for thread-specific data are then called for all keys that have non-NULL values associated with them in the calling thread (see <b>pthread_key_create()</b> for more details). Finally, execution of the calling thread is stopped.</p> <p>The <i>status</i> argument is the return value of the thread and can be consulted from another thread using <b>pthread_join()</b> unless this thread was detached (<i>i.e.</i>, a call to <b>pthread_detach()</b> had been made for it, or it was created in the detached state).</p> <p>All threads that remain <i>joinable</i> at the time they exit should ensure that <b>pthread_join()</b> is called on their behalf by another thread to reclaim the resources that they hold.</p> |
| <b>RETURNS</b>     | Does not return.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |

ERRNOS N/A

SEE ALSO *pthreadLib*, *pthread\_cleanup\_push()*, *pthread\_detach()*, *pthread\_join()*, *pthread\_key\_create()*

---

## pthread\_getschedparam()

NAME *pthread\_getschedparam()* – get value of schedparam attribute from a thread (POSIX)

SYNOPSIS

```
int pthread_getschedparam
(
    pthread_t      thread, /* thread */
    int *          pPolicy, /* current policy (out) */
    struct sched_param * pParam /* current parameters (out) */
)
```

DESCRIPTION This routine reads the current scheduling parameters and policy of the thread specified by *thread*. The information is returned via *pPolicy* and *pParam*.

RETURNS On success zero; on failure a non-zero error code.

ERRNOS ESRCH

SEE ALSO *pthreadLib*, *schedPxLib*, *pthread\_attr\_getschedparam()*, *pthread\_attr\_getschedpolicy()*, *pthread\_attr\_setschedparam()*, *pthread\_attr\_setschedpolicy()*, *pthread\_setschedparam()*, *sched\_getparam()*, *sched\_setparam()*

---

## pthread\_getspecific()

NAME *pthread\_getspecific()* – get thread specific data (POSIX)

SYNOPSIS

```
void *pthread_getspecific
(
    pthread_key_t key /* thread specific data key */
)
```

|                    |                                                                                                                |
|--------------------|----------------------------------------------------------------------------------------------------------------|
| <b>DESCRIPTION</b> | This routine returns the value associated with the thread specific data key <i>key</i> for the calling thread. |
| <b>RETURNS</b>     | The value associated with <i>key</i> , or NULL.                                                                |
| <b>ERRNOS</b>      | N/A                                                                                                            |
| <b>SEE ALSO</b>    | <b>pthreadLib</b> , <b>pthread_key_create()</b> , <b>pthread_key_delete()</b> , <b>pthread_setspecific()</b>   |

---

## pthread\_join()

**NAME** pthread\_join() – wait for a thread to terminate (POSIX)

**SYNOPSIS**

```
int pthread_join
(
    pthread_t thread,          /* thread to wait for */
    void *   *ppStatus        /* exit status of thread (out) */
)
```

**DESCRIPTION** This routine will block the calling thread until the thread specified by *thread* terminates, or is canceled. The thread must be in the joinable state, *i.e.*, it cannot have been detached by a call to **pthread\_detach()**, or created in the detached state.

If *ppStatus* is not NULL, when *thread* terminates its exit status will be stored in the specified location. The exit status will be either the value passed to **pthread\_exit()**, or **PTHREAD\_CANCELED** if the thread was canceled.

Only one thread can wait for the termination of a given thread. If another thread is already waiting when this function is called an error will be returned (EINVAL).

If the calling thread passes its own ID in *thread*, the call will fail with the error **EDEADLK**.

---

**NOTE:** All threads that remain *joinable* at the time they exit should ensure that **pthread\_join()** is called on their behalf by another thread to reclaim the resources that they hold.

---

**RETURNS** On success zero; on failure a non-zero error code.

**ERRNOS** EINVAL, ESRCH, EDEADLK

**SEE ALSO** **pthreadLib**, **pthread\_detach()**, **pthread\_exit()**

---

## pthread\_key\_create()

**NAME** pthread\_key\_create() – create a thread specific data key (POSIX)

**SYNOPSIS**

```
int pthread_key_create
(
    pthread_key_t * pKey,          /* thread specific data key */
    void (* destructor)(void * ) /* destructor function */
)
```

**DESCRIPTION** This routine allocates a new thread specific data key. The key is stored in the location pointed to by *key*. The value initially associated with the returned key is **NULL** in all currently executing threads. If the maximum number of keys are already allocated, the function returns an error (**EAGAIN**).

The *destructor* parameter specifies a destructor function associated with the key. When a thread terminates via **pthread\_exit()**, or by cancellation, *destructor* is called with the value associated with the key in that thread as an argument. The destructor function is **not** called if that value is **NULL**. The order in which destructor functions are called at thread termination time is unspecified.

**RETURNS** On success zero; on failure a non-zero error code.

**ERRNOS** EAGAIN

**SEE ALSO** pthreadLib, pthread\_getspecific(), pthread\_key\_delete(), pthread\_setspecific()

---

## pthread\_key\_delete()

**NAME** pthread\_key\_delete() – delete a thread specific data key (POSIX)

**SYNOPSIS**

```
int pthread_key_delete
(
    pthread_key_t key              /* thread specific data key to delete */
)
```

**DESCRIPTION** This routine deletes the thread specific data associated with *key*, and deallocates the key itself. It does not call any destructor associated with the key.

**RETURNS** On success zero; on failure a non-zero error code.

ERRNOS EINVAL

SEE ALSO pthreadLib, pthread\_key\_create()

---

## pthread\_kill()

**NAME** pthread\_kill() – send a signal to a thread (POSIX)

**SYNOPSIS**

```
int pthread_kill
(
    pthread_t thread,      /* thread to signal */
    int sig               /* signal to send */
)
```

**DESCRIPTION** This routine sends signal number *sig* to the thread specified by *thread*. The signal is delivered and handled as described for the `kill()` function.

**RETURNS** On success zero; on failure a non-zero error code.

**ERRNOS** ESRCH, EINVAL

**SEE ALSO** pthreadLib, kill(), pthread\_sigmask(), sigprocmask(), sigaction(), sigsuspend(), sigwait()

---

## pthread\_mutex\_destroy()

**NAME** pthread\_mutex\_destroy() – destroy a mutex (POSIX)

**SYNOPSIS**

```
int pthread_mutex_destroy
(
    pthread_mutex_t * pMutex /* POSIX mutex */
)
```

**DESCRIPTION** This routine destroys a mutex object, freeing the resources it might hold. The mutex must be unlocked when this function is called, otherwise it will return an error (EBUSY).

**RETURNS** On success zero; on failure a non-zero error code.

**ERRNOS** EINVAL, EBUSY

**SEE ALSO** pthreadLib, semLib, semMLib, pthread\_mutex\_init(), pthread\_mutex\_lock(), pthread\_mutex\_trylock(), pthread\_mutex\_unlock(), pthread\_mutexattr\_init(), semDelete()

---

## pthread\_mutex\_getprioceiling()

**NAME** pthread\_mutex\_getprioceiling() – get the value of the prioceiling attribute of a mutex (POSIX)

**SYNOPSIS**

```
int pthread_mutex_getprioceiling
(
    pthread_mutex_t * pMutex,      /* POSIX mutex */
    int *           pPrioceiling /* current priority ceiling (out) */
)
```

**DESCRIPTION** This function gets the current value of the prioceiling attribute of a mutex. Unless the mutex was created with a protocol attribute value of PTHREAD\_PRIO\_PROTECT, this value is meaningless.

**RETURNS** On success zero; on failure a non-zero error code.

**ERRNOS** EINVAL

**SEE ALSO** pthreadLib, pthread\_mutex\_setprioceiling(), pthread\_mutexattr\_getprioceiling(), pthread\_mutexattr\_setprioceiling()

---

## pthread\_mutex\_init()

**NAME** pthread\_mutex\_init() – initialize mutex from attributes object (POSIX)

**SYNOPSIS**

```
int pthread_mutex_init
(
    pthread_mutex_t * pMutex, /* POSIX mutex */
    const pthread_mutexattr_t * pAttr /* mutex attributes */
)
```

**DESCRIPTION** This routine initializes the mutex object pointed to by *pMutex* according to the mutex attributes specified in *pAttr*. If *pAttr* is **NULL**, default attributes are used as defined in the POSIX specification.

**RETURNS** On success zero; on failure a non-zero error code.

**ERRNOS** EINVAL, EBUSY

**SEE ALSO** pthreadLib, semLib, semMLib, pthread\_mutex\_destroy(), pthread\_mutex\_lock(), pthread\_mutex\_trylock(), pthread\_mutex\_unlock(), pthread\_mutexattr\_init(), semMCreate()

---

## pthread\_mutex\_lock()

**NAME** pthread\_mutex\_lock() – lock a mutex (POSIX)

**SYNOPSIS**

```
int pthread_mutex_lock
(
    pthread_mutex_t * pMutex /* POSIX mutex */
)
```

**DESCRIPTION** This routine locks the mutex specified by *pMutex*. If the mutex is currently unlocked, it becomes locked, and is said to be owned by the calling thread. In this case **pthread\_mutex\_lock()** returns immediately.

If the mutex is already locked by another thread, **pthread\_mutex\_lock()** blocks the calling thread until the mutex is unlocked by its current owner.

If it is already locked by the calling thread, **pthread\_mutex\_lock** will deadlock on itself and the thread will block indefinitely.

**RETURNS** On success zero; on failure a non-zero error code.

**ERRNOS** EINVAL

**SEE ALSO** pthreadLib, semLib, semMLib, pthread\_mutex\_init(), pthread\_mutex\_lock(), pthread\_mutex\_trylock(), pthread\_mutex\_unlock(), pthread\_mutexattr\_init(), semTake()

---

## pthread\_mutex\_setprioceiling()

**NAME** pthread\_mutex\_setprioceiling() – dynamically set the prioceiling attribute of a mutex (POSIX)

**SYNOPSIS**

```
int pthread_mutex_setprioceiling
(
    pthread_mutex_t * pMutex,          /* POSIX mutex */
    int               prioceiling,     /* new priority ceiling */
    int *             pOldPrioceiling /* old priority ceiling (out) */
)
```

**DESCRIPTION** This function dynamically sets the value of the prioceiling attribute of a mutex. Unless the mutex was created with a protocol value of `PTHREAD_PRIO_PROTECT`, this function does nothing.

**RETURNS** On success zero; on failure a non-zero error code.

**ERRNOS** EINVAL, EPERM, S\_objLib\_OBJ\_ID\_ERROR, S\_semLib\_NOT\_ISR\_CALLABLE

**SEE ALSO** pthreadLib, pthread\_mutex\_getprioceiling(), pthread\_mutexattr\_getprioceiling(), pthread\_mutexattr\_setprioceiling()

---

## pthread\_mutex\_trylock()

**NAME** pthread\_mutex\_trylock() – lock mutex if it is available (POSIX)

**SYNOPSIS**

```
int pthread_mutex_trylock
(
    pthread_mutex_t * pMutex /* POSIX mutex */
)
```

**DESCRIPTION** This routine locks the mutex specified by *pMutex*. If the mutex is currently unlocked, it becomes locked and owned by the calling thread. In this case `pthread_mutex_trylock()` returns immediately.

If the mutex is already locked by another thread, `pthread_mutex_trylock()` returns immediately with the error code `EBUSY`.

**RETURNS** On success zero; on failure a non-zero error code.

***pthread\_mutex\_unlock()***

**ERRNOS** EINVAL, EBUSY

**SEE ALSO** pthreadLib, semLib, semMLib, pthread\_mutex\_init(), pthread\_mutex\_lock(), pthread\_mutex\_trylock(), pthread\_mutex\_unlock(), pthread\_mutexattr\_init(), semTake()

---

## pthread\_mutex\_unlock()

**NAME** pthread\_mutex\_unlock() – unlock a mutex (POSIX)

**SYNOPSIS**

```
int pthread_mutex_unlock
(
    pthread_mutex_t * pMutex
)
```

**DESCRIPTION** This routine unlocks the mutex specified by *pMutex*. If the calling thread is not the current owner of the mutex, **pthread\_mutex\_unlock()** returns with the error code **EPERM**.

**RETURNS** On success zero; on failure a non-zero error code.

**ERRNOS** EINVAL, EPERM, S\_objLib\_OBJ\_ID\_ERROR, S\_semLib\_NOT\_ISR\_CALLABLE

**SEE ALSO** pthreadLib, semLib, semMLib, pthread\_mutex\_init(), pthread\_mutex\_lock(), pthread\_mutex\_trylock(), pthread\_mutex\_unlock(), pthread\_mutexattr\_init(), semGive()

---

## pthread\_mutexattr\_destroy()

**NAME** pthread\_mutexattr\_destroy() – destroy mutex attributes object (POSIX)

**SYNOPSIS**

```
int pthread_mutexattr_destroy
(
    pthread_mutexattr_t * pAttr /* mutex attributes */
)
```

**DESCRIPTION** This routine destroys a mutex attribute object. The mutex attribute object must not be reused until it is re-initialized.

**RETURNS** On success zero; on failure a non-zero error code.

**ERRNOS** EINVAL

**SEE ALSO** pthreadLib, pthread\_mutexattr\_getprioceiling(), pthread\_mutexattr\_getprotocol(), pthread\_mutexattr\_init(), pthread\_mutexattr\_setprioceiling(), pthread\_mutexattr\_setprotocol(), pthread\_mutex\_init()

---

## pthread\_mutexattr\_getprioceiling()

**NAME** pthread\_mutexattr\_getprioceiling() – get the current value of the prioceiling attribute in a mutex attributes object (POSIX)

**SYNOPSIS**

```
int pthread_mutexattr_getprioceiling
(
    pthread_mutexattr_t * pAttr,          /* mutex attributes */
    int * pPrioceiling /* current priority ceiling (out) */
)
```

**DESCRIPTION** This function gets the current value of the prioceiling attribute in a mutex attributes object. Unless the value of the protocol attribute is PTHREAD\_PRIO\_PROTECT, this value is ignored.

**RETURNS** On success zero; on failure a non-zero error code.

**ERRNOS** EINVAL

**SEE ALSO** pthreadLib, pthread\_mutexattr\_destroy(), pthread\_mutexattr\_getprotocol(), pthread\_mutexattr\_init(), pthread\_mutexattr\_setprioceiling(), pthread\_mutexattr\_setprotocol(), pthread\_mutex\_init()

---

## pthread\_mutexattr\_getprotocol()

**NAME** pthread\_mutexattr\_getprotocol() – get value of protocol in mutex attributes object (POSIX)

**SYNOPSIS**

```
int pthread_mutexattr_getprotocol
(
    pthread_mutexattr_t * pAttr, /* mutex attributes */
    int *                pProtocol /* current protocol (out) */
)
```

**DESCRIPTION** This function gets the current value of the protocol attribute in a mutex attributes object.

**RETURNS** On success zero; on failure a non-zero error code.

**ERRNOS** EINVAL

**SEE ALSO** pthreadLib, pthread\_mutexattr\_destroy(), pthread\_mutexattr\_getprioceiling(), pthread\_mutexattr\_init(), pthread\_mutexattr\_setprioceiling(), pthread\_mutexattr\_setprotocol(), pthread\_mutex\_init()

---

## pthread\_mutexattr\_init()

**NAME** pthread\_mutexattr\_init() – initialize mutex attributes object (POSIX)

**SYNOPSIS**

```
int pthread_mutexattr_init
(
    pthread_mutexattr_t * pAttr /* mutex attributes */
)
```

**DESCRIPTION** This routine initializes the mutex attribute object *pAttr* and fills it with default values for the attributes as defined by the POSIX specification.

**RETURNS** On success zero; on failure a non-zero error code.

**ERRNOS** EINVAL

**SEE ALSO** pthreadLib, pthread\_mutexattr\_destroy(), pthread\_mutexattr\_getprioceiling(), pthread\_mutexattr\_getprotocol(), pthread\_mutexattr\_setprioceiling(), pthread\_mutexattr\_setprotocol(), pthread\_mutex\_init()

---

## pthread\_mutexattr\_setprioceiling()

**NAME** pthread\_mutexattr\_setprioceiling() – set prioceiling attribute in mutex attributes object (POSIX)

**SYNOPSIS**

```
int pthread_mutexattr_setprioceiling
(
    pthread_mutexattr_t * pAttr,      /* mutex attributes */
    int                  prioceiling /* new priority ceiling */
)
```

**DESCRIPTION** This function sets the value of the prioceiling attribute in a mutex attributes object. Unless the protocol attribute is set to PTHREAD\_PRIO\_PROTECT, this attribute is ignored.

**RETURNS** On success zero; on failure a non-zero error code.

**ERRNOS** EINVAL

**SEE ALSO** pthreadLib, pthread\_mutexattr\_destroy(), pthread\_mutexattr\_getprioceiling(), pthread\_mutexattr\_getprotocol(), pthread\_mutexattr\_init(), pthread\_mutexattr\_setprotocol(), pthread\_mutex\_init()

---

## pthread\_mutexattr\_setprotocol()

**NAME** pthread\_mutexattr\_setprotocol() – set protocol attribute in mutex attribute object (POSIX)

**SYNOPSIS**

```
int pthread_mutexattr_setprotocol
(
    pthread_mutexattr_t * pAttr,      /* mutex attributes */
    int                  protocol     /* new protocol */
)
```

**DESCRIPTION** This function selects the locking protocol to be used when a mutex is created using this attributes object. The protocol to be selected is either PTHREAD\_PRIO\_INHERIT or PTHREAD\_PRIO\_PROTECT.

**RETURNS** On success zero; on failure a non-zero error code.

**ERRNOS** EINVAL, ENOTSUP

SEE ALSO **pthreadLib**, **pthread\_mutexattr\_destroy()**, **pthread\_mutexattr\_getprioceiling()**,  
**pthread\_mutexattr\_getprotocol()**, **pthread\_mutexattr\_init()**,  
**pthread\_mutexattr\_setprioceiling()**, **pthread\_mutex\_init()**

---

## pthread\_once()

NAME **pthread\_once()** – dynamic package initialization (POSIX)

SYNOPSIS

```
int pthread_once
(
    pthread_once_t * onceControl, /* once control location */
    void (* initFunc)(void)      /* function to call */
)
```

DESCRIPTION This routine provides a mechanism to ensure that one, and only one call to a user specified initialization function will occur. This allows all threads in a system to attempt initialization of some feature they need to use, without any need for the application to explicitly prevent multiple calls.

When a thread makes a call to **pthread\_once()**, the first thread to call it with the specified control variable, *onceControl*, will result in a call to *initFunc*, but subsequent calls will not. The *onceControl* parameter determines whether the associated initialization routine has been called. The *initFunc* function is complete when **pthread\_once()** returns.

The function **pthread\_once()** is not a cancellation point; however, if the function *initFunc* is a cancellation point, and the thread is canceled while executing it, the effect on *onceControl* is the same as if **pthread\_once()** had never been called.

---

**WARNING:** If *onceControl* has automatic storage duration or is not initialized to the value **PTHREAD\_ONCE\_INIT**, the behavior of **pthread\_once()** is undefined. The constant **PTHREAD\_ONCE\_INIT** is defined in the **pthread.h** header file.

---

RETURNS Always returns zero.

ERRNOS None.

SEE ALSO **pthreadLib**

---

## pthread\_self()

**NAME** `pthread_self()` – get the calling thread’s ID (POSIX)

**SYNOPSIS** `pthread_t pthread_self (void)`

**DESCRIPTION** This function returns the calling thread’s ID.

**RETURNS** Calling thread’s ID.

**SEE ALSO** `pthreadLib`

---

## pthread\_setcancelstate()

**NAME** `pthread_setcancelstate()` – set cancellation state for calling thread (POSIX)

**SYNOPSIS**

```
int pthread_setcancelstate
(
    int state,                /* new state */
    int * oldstate           /* old state (out) */
)
```

**DESCRIPTION** This routine sets the cancellation state for the calling thread to *state*, and, if *oldstate* is not `NULL`, returns the old state in the location pointed to by *oldstate*.

The state can be one of the following:

**PTHREAD\_CANCEL\_ENABLE**  
Enable thread cancellation.

**PTHREAD\_CANCEL\_DISABLE**  
Disable thread cancellation (*i.e.*, thread cancellation requests are ignored).

**RETURNS** On success zero; on failure a non-zero error code.

**ERRNOS** `EINVAL`

**SEE ALSO** `pthreadLib`, `pthread_cancel()`, `pthread_setcanceltype()`, `pthread_testcancel()`

## **pthread\_setcanceltype()**

- NAME** `pthread_setcanceltype()` – set cancellation type for calling thread (POSIX)
- SYNOPSIS**
- ```
int pthread_setcanceltype
(
    int  type,                /* new type */
    int * oldtype             /* old type (out) */
)
```
- DESCRIPTION**
- This routine sets the cancellation type for the calling thread to *type*. If *oldtype* is not NULL, then the old cancellation type is stored in the location pointed to by *oldtype*.
- Possible values for *type* are:
- PTHREAD\_CANCEL\_ASYNCHRONOUS**  
Any cancellation request received by this thread will be acted upon as soon as it is received.
- PTHREAD\_CANCEL\_DEFERRED**  
Cancellation requests received by this thread will be deferred until the next cancellation point is reached.
- RETURNS** On success zero; on failure a non-zero error code.
- ERRNOS** EINVAL
- SEE ALSO** `pthreadLib`, `pthread_cancel()`, `pthread_setcancelstate()`, `pthread_testcancel()`
- 

## **pthread\_setschedparam()**

- NAME** `pthread_setschedparam()` – dynamically set schedparam attribute for a thread (POSIX)
- SYNOPSIS**
- ```
int pthread_setschedparam
(
    pthread_t      thread, /* thread */
    int            policy, /* new policy */
    const struct sched_param * pParam /* new parameters */
)
```

**DESCRIPTION** This routine will set the scheduling parameters (*pParam*) and policy (*policy*) for the thread specified by *thread*.

In VxWorks the scheduling policy is global and not set on a per-thread basis; if the selected policy does not match the current global setting then this function will return an error (EINVAL).

**RETURNS** On success zero; on failure a non-zero error code.

**ERRNOS** EINVAL, ESRCH

**SEE ALSO** pthreadLib, schedPxLib, pthread\_attr\_getschedparam(), pthread\_attr\_getschedpolicy(), pthread\_attr\_setschedparam(), pthread\_attr\_setschedpolicy(), pthread\_getschedparam(), sched\_getparam(), sched\_setparam()

---

## pthread\_setspecific()

**NAME** pthread\_setspecific() – set thread specific data (POSIX)

**SYNOPSIS**

```
int pthread_setspecific
(
    pthread_key_t key,          /* thread specific data key */
    const void * value         /* new value */
)
```

**DESCRIPTION** Sets the value of the thread specific data associated with *key* to *value* for the calling thread.

**RETURNS** On success zero; on failure a non-zero error code.

**ERRNOS** EINVAL, ENOMEM

**SEE ALSO** pthreadLib, pthread\_getspecific(), pthread\_key\_create(), pthread\_key\_delete()

## pthread\_sigmask()

**NAME** pthread\_sigmask() – change and/or examine calling thread’s signal mask (POSIX)

**SYNOPSIS**

```
int pthread_sigmask
(
    int          how,          /* method for changing set */
    const sigset_t * set,     /* new set of signals */
    sigset_t *   oset        /* old set of signals */
)
```

**DESCRIPTION** This routine changes the signal mask for the calling thread as described by the *how* and *set* arguments. If *oset* is not NULL, the previous signal mask is stored in the location pointed to by it.

The value of *how* indicates the manner in which the set is changed and consists of one of the following defined in **signal.h**:

**SIG\_BLOCK**

The resulting set is the union of the current set and the signal set pointed to by *set*.

**SIG\_UNBLOCK**

The resulting set is the intersection of the current set and the complement of the signal set pointed to by *set*.

**SIG\_SETMASK**

The resulting set is the signal set pointed to by *oset*.

**RETURNS** On success zero; on failure a non-zero error code is returned.

**ERRNOS** EINVAL

**SEE ALSO** pthreadLib, kill(), pthread\_kill(), sigprocmask(), sigaction(), sigsuspend(), sigwait()

---

## pthread\_testcancel()

|                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
|--------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>        | <code>pthread_testcancel()</code> – create a cancellation point in the calling thread (POSIX)                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| <b>SYNOPSIS</b>    | <code>void pthread_testcancel (void)</code>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| <b>DESCRIPTION</b> | <p>This routine creates a cancellation point in the calling thread. It has no effect if cancellation is disabled (<i>i.e.</i>, the cancellation state has been set to <code>PTHREAD_CANCEL_DISABLE</code> using the <code>pthread_setcancelstate()</code> function).</p> <p>If cancellation is enabled, the cancellation type is <code>PTHREAD_CANCEL_DEFERRED</code> and a cancellation request has been received, then this routine will call <code>pthread_exit()</code> with the exit status set to <code>PTHREAD_CANCELED</code>. If any of these conditions is not met, then the routine does nothing.</p> |
| <b>RETURNS</b>     | N/A                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| <b>ERRNOS</b>      | N/A                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| <b>SEE ALSO</b>    | <code>pthreadLib</code> , <code>pthread_cancel()</code> , <code>pthread_setcancelstate()</code> , <code>pthread_setcanceltype()</code>                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |

---

## ptyDevCreate()

|                    |                                                                                                                                                                                                                                                                                                                                                                   |
|--------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>        | <code>ptyDevCreate()</code> – create a pseudo terminal                                                                                                                                                                                                                                                                                                            |
| <b>SYNOPSIS</b>    | <pre>STATUS ptyDevCreate (     char * name,           /* name of pseudo terminal */     int   rdBufSize,      /* size of terminal read buffer */     int   wrtBufSize      /* size of write buffer */ )</pre>                                                                                                                                                     |
| <b>DESCRIPTION</b> | <p>This routine creates a master and slave device which can then be opened by the master and slave processes. The master process simulates the “hardware” side of the driver, while the slave process is the application program that normally talks to a tty driver. Data written to the master device can then be read on the slave device, and vice versa.</p> |
| <b>RETURNS</b>     | OK, or ERROR if memory is insufficient.                                                                                                                                                                                                                                                                                                                           |
| <b>SEE ALSO</b>    | <code>ptyDrv</code>                                                                                                                                                                                                                                                                                                                                               |

## ptyDevRemove()

|                    |                                                                                                                                               |
|--------------------|-----------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>        | ptyDevRemove() – destroy a pseudo terminal                                                                                                    |
| <b>SYNOPSIS</b>    | <pre>STATUS ptyDevRemove (     char * pName          /* name of pseudo terminal to remove */ )</pre>                                          |
| <b>DESCRIPTION</b> | This routine removes an existing master and slave device and releases all allocated memory. It will close any open files using either device. |
| <b>RETURNS</b>     | OK, or ERROR if terminal not found                                                                                                            |
| <b>SEE ALSO</b>    | ptyDrv                                                                                                                                        |

---

## ptyDrv()

|                    |                                                                                                                 |
|--------------------|-----------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>        | ptyDrv() – initialize the pseudo-terminal driver                                                                |
| <b>SYNOPSIS</b>    | <pre>STATUS ptyDrv (void)</pre>                                                                                 |
| <b>DESCRIPTION</b> | This routine initializes the pseudo-terminal driver. It must be called before any other routine in this module. |
| <b>RETURNS</b>     | OK, or ERROR if the master or slave devices cannot be installed.                                                |
| <b>SEE ALSO</b>    | ptyDrv                                                                                                          |

---

## ptyShow()

**NAME** `ptyShow()` – show the state of the Pty Buffers

**SYNOPSIS** `void ptyShow (void)`

**SEE ALSO** `ptyDrv`

---

## putc()

**NAME** `putc()` – write a character to a stream (ANSI)

**SYNOPSIS**

```
int putc
(
    int    c,                /* character to write */
    FILE * fp                /* stream to write to */
)
```

**DESCRIPTION** This routine writes a character *c* to a specified stream, at the position indicated by the stream's file position indicator (if defined), and advances the indicator appropriately.

This routine is equivalent to `fputc()`, except that if it is implemented as a macro, it may evaluate *fp* more than once; thus, the argument should never be an expression with side effects.

**INCLUDE FILES** `stdio.h`

**RETURNS** The character written, or EOF if a write error occurs, with the error indicator set for the stream.

**SEE ALSO** `ansiStdio, fputc()`

## putchar()

|                      |                                                                                                                                                                                                                                                                                                      |
|----------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>          | <b>putchar()</b> – write a character to the standard output stream (ANSI)                                                                                                                                                                                                                            |
| <b>SYNOPSIS</b>      | <pre>int putchar (     int c                /* character to write */ )</pre>                                                                                                                                                                                                                         |
| <b>DESCRIPTION</b>   | <p>This routine writes a character <i>c</i> to the standard output stream, at the position indicated by the stream's file position indicator (if defined), and advances the indicator appropriately.</p> <p>This routine is equivalent to <b>putc()</b> with a second argument of <b>stdout</b>.</p> |
| <b>INCLUDE FILES</b> | <b>stdio.h</b>                                                                                                                                                                                                                                                                                       |
| <b>RETURNS</b>       | The character written, or EOF if a write error occurs, with the error indicator set for the standard output stream.                                                                                                                                                                                  |
| <b>SEE ALSO</b>      | <b>ansiStdio, putc(), fputc()</b>                                                                                                                                                                                                                                                                    |

---

## putenv()

|                    |                                                                                                                                                                                                                                                                                        |
|--------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>        | <b>putenv()</b> – set an environment variable                                                                                                                                                                                                                                          |
| <b>SYNOPSIS</b>    | <pre>STATUS putenv (     char * pEnvString    /* string to add to env */ )</pre>                                                                                                                                                                                                       |
| <b>DESCRIPTION</b> | <p>This routine sets an environment variable to a value by altering an existing variable or creating a new one. The parameter points to a string of the form “variableName=value”. Unlike the UNIX implementation, the string passed as a parameter is copied to a private buffer.</p> |
| <b>RETURNS</b>     | OK, or ERROR if space cannot be malloc'd.                                                                                                                                                                                                                                              |
| <b>SEE ALSO</b>    | <b>envLib, envLibInit(), getenv()</b>                                                                                                                                                                                                                                                  |

---

## puts()

|                      |                                                                                                                                                                       |
|----------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>          | <b>puts()</b> – write a string to the standard output stream (ANSI)                                                                                                   |
| <b>SYNOPSIS</b>      | <pre>int puts (     char const * s          /* string to write */ )</pre>                                                                                             |
| <b>DESCRIPTION</b>   | This routine writes to the standard output stream a specified string <i>s</i> , minus the terminating null character, and appends a new-line character to the output. |
| <b>INCLUDE FILES</b> | <b>stdio.h</b>                                                                                                                                                        |
| <b>RETURNS</b>       | A non-negative value, or EOF if a write error occurs.                                                                                                                 |
| <b>SEE ALSO</b>      | <b>ansiStdio, fputs()</b>                                                                                                                                             |

---

## putw()

|                      |                                                                                                                                                       |
|----------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>          | <b>putw()</b> – write a word (32-bit integer) to a stream                                                                                             |
| <b>SYNOPSIS</b>      | <pre>int putw (     int    w,              /* word (32-bit integer) */     FILE * fp              /* output stream */ )</pre>                         |
| <b>DESCRIPTION</b>   | This routine appends the 32-bit quantity <i>w</i> to a specified stream.<br>This routine is provided for compatibility with earlier VxWorks releases. |
| <b>INCLUDE FILES</b> | <b>stdio.h</b>                                                                                                                                        |
| <b>RETURNS</b>       | The value written.                                                                                                                                    |
| <b>SEE ALSO</b>      | <b>ansiStdio</b>                                                                                                                                      |

***pwd()***

---

## **pwd()**

**NAME** `pwd()` – print the current default directory

**SYNOPSIS** `void pwd (void)`

**DESCRIPTION** This command displays the current working device/directory.

---

**NOTE:** This is a target resident function, which manipulates the target I/O system. It must be preceded with the @ letter if executed from the Tornado Shell (**windsh**), which has a built-in command of the same name that operates on the Host's I/O system.

---

**RETURNS** N/A

**SEE ALSO** `usrFsLib, cd()`, *VxWorks Programmer's Guide: Target Shell*, **windsh**, *Tornado User's Guide: Shell*

---

## qsort()

**NAME** `qsort()` – sort an array of objects (ANSI)

**SYNOPSIS**

```
void qsort
(
    void * bot,           /* initial element in array */
    size_t nmemb,        /* no. of objects in array */
    size_t size,         /* size of array element */
    int (* compar) (const void * , const void * )
                        /* comparison function */
)
```

**DESCRIPTION** This routine sorts an array of *nmemb* objects, the initial element of which is pointed to by *bot*. The size of each object is specified by *size*.

The contents of the array are sorted into ascending order according to a comparison function pointed to by *compar*, which is called with two arguments that point to the objects being compared. The function shall return an integer less than, equal to, or greater than zero if the first argument is considered to be respectively less than, equal to, or greater than the second.

If two elements compare as equal, their order in the sorted array is unspecified.

**INCLUDE FILES** `stdlib.h`

**RETURNS** N/A

**SEE ALSO** `ansiStdlib`

**r0()**

---

**NAME** **r0()** – return the contents of register **r0** (also **r1 - r14**, **r1-r15** for SH) (ARM, SH)

**SYNOPSIS**

```
int r0
(
    int taskId          /* task ID, 0 means default task */
)
```

**DESCRIPTION** This command extracts the contents of register **r0** from the TCB of a specified task. If *taskId* is omitted or zero, the last task referenced is assumed.

Similar routines are provided for registers (**r1 - r15**): **r1()** - **r15()**.

**RETURNS** The contents of register **r0** (or the requested register).

**SEE ALSO** **dbgArchLib**, *VxWorks Programmer's Guide: Debugging*

---

**raise()**

**NAME** **raise()** – send a signal to the caller's task

**SYNOPSIS**

```
int raise
(
    int signo          /* signal to send to caller's task */
)
```

**DESCRIPTION** This routine sends the signal *signo* to the task invoking the call.

**RETURNS** **OK** (0), or **ERROR** (-1) if the signal number or task ID is invalid.

**ERRNO** **EINVAL**

**SEE ALSO** **sigLib**

---

## ramDevCreate()

**NAME** `ramDevCreate()` – create a RAM disk device

**SYNOPSIS**

```
BLK_DEV *ramDevCreate
(
    char * ramAddr,          /* where it is in memory (0 = malloc) */
    int   bytesPerBlk,      /* number of bytes per block */
    int   blksPerTrack,     /* number of blocks per track */
    int   nBlocks,         /* number of blocks on this device */
    int   blkOffset        /* no. of blks to skip at start of device */
)
```

**DESCRIPTION** This routine creates a RAM disk device.

Memory for the RAM disk can be pre-allocated separately; if so, the *ramAddr* parameter should be the address of the pre-allocated device memory. Or, memory can be automatically allocated with `malloc()` by setting *ramAddr* to zero.

The *bytesPerBlk* parameter specifies the size of each logical block on the RAM disk. If *bytesPerBlk* is zero, 512 is used.

The *blksPerTrack* parameter specifies the number of blocks on each logical track of the RAM disk. If *blksPerTrack* is zero, the count of blocks per track is set to *nBlocks* (i.e., the disk is defined as having only one track).

The *nBlocks* parameter specifies the size of the disk, in blocks. If *nBlocks* is zero, a default size is used. The default is calculated using a total disk size of either 51,200 bytes or one-half of the size of the largest memory area available, whichever is less. This default disk size is then divided by *bytesPerBlk* to determine the number of blocks.

The *blkOffset* parameter specifies an offset, in blocks, from the start of the device to be used when writing or reading the RAM disk. This offset is added to the block numbers passed by the file system during disk accesses. (VxWorks file systems always use block numbers beginning at zero for the start of a device.) This offset value is typically useful only if a specific address is given for *ramAddr*. Normally, *blkOffset* is 0.

**FILE SYSTEMS** Once the device has been created, it must be associated with a name and a file system (dosFs, rt11Fs, or rawFs). This is accomplished using the file system's device initialization routine or make-file-system routine, e.g., `dosFsDevInit()` or `dosFsMkfs()`. The `ramDevCreate()` call returns a pointer to a block device structure (`BLK_DEV`). This structure contains fields that describe the physical properties of a disk device and specify the addresses of routines within the `ramDrv` driver. The `BLK_DEV` structure address must be passed to the desired file system (dosFs, rt11Fs or rawFs) via the file system's device initialization or make-file-system routine. Only then is a name and file system associated with the device, making it available for use.

**ramDiskDevCreate()****EXAMPLE**

In the following example, a 200-Kbyte RAM disk is created with automatically allocated memory, 512-byte blocks, a single track, and no block offset. The device is then initialized for use with dosFs and assigned the name "DEV1:":

```
BLK_DEV *pBlkDev;
DOS_VOL_DESC *pVolDesc;
pBlkDev = ramDevCreate (0, 512, 400, 400, 0);
pVolDesc = dosFsMkfs ("DEV1:", pBlkDev);
```

The `dosFsMkfs()` routine calls `dosFsDevInit()` with default parameters and initializes the file system on the disk by calling `ioctl()` with the `FIODISKINIT` function.

If the RAM disk memory already contains a disk image created elsewhere, the first argument to `ramDevCreate()` should be the address in memory, and the formatting parameters -- *bytesPerBlk*, *blksPerTrack*, *nBlocks*, and *blkOffset* -- must be identical to those used when the image was created. For example:

```
pBlkDev = ramDevCreate (0xc0000, 512, 400, 400, 0);
pVolDesc = dosFsDevInit ("DEV1:", pBlkDev, NULL);
```

In this case, `dosFsDevInit()` must be used instead of `dosFsMkfs()`, because the file system already exists on the disk and should not be re-initialized. This procedure is useful if a RAM disk is to be created at the same address used in a previous boot of VxWorks. The contents of the RAM disk will then be preserved.

These same procedures apply when creating a RAM disk with `rt11Fs` using `rt11FsDevInit()` and `rt11FsMkfs()`, or creating a RAM disk with `rawFs` using `rawFsDevInit()`.

**RETURNS**

A pointer to a block device structure (`BLK_DEV`) or `NULL` if memory cannot be allocated for the device structure or for the RAM disk.

**SEE ALSO**

`ramDrv`, `dosFsMkfs()`, `dosFsDevInit()`, `rt11FsDevInit()`, `rt11FsMkfs()`, `rawFsDevInit()`

---

## ramDiskDevCreate()

**NAME**

`ramDiskDevCreate()` – initialize a RAM Disk device

**SYNOPSIS**

```
CBIO_DEV_ID ramDiskDevCreate
(
    char * pRamAddr,          /* where it is in memory (0 = malloc) */
    int   bytesPerBlk,        /* number of bytes per block */
    int   blksPerTrack,       /* number of blocks per track */
    int   nBlocks,           /* number of blocks on this device */

```

```
int    blkOffset    /* no. of blks to skip at start of device */
)
```

**DESCRIPTION** This function creates a compact RAM-Disk device that can be directly utilized by **dosFsLib**, without the intermediate disk cache. It can be used for non-volatile RAM as well as volatile RAM disks.

The RAM size is specified in terms of total number of blocks in the device and the block size in bytes. The minimal block size is 32 bytes. If *pRamAddr* is NULL, space will be allocated from the default memory pool.

**RETURNS** a CPIO handle that can be directly used by **dosFsDevCreate()** or NULL if the requested amount of RAM is not available.

---

**WARNING:** When used with NV-RAM, this module can not eliminate mid-block write interruption, which may cause file system corruption not existent in common disk drives.

---

**SEE ALSO** **ramDiskCbio**, **dosFsDevCreate()**.

---

## ramDrv()

**NAME** **ramDrv()** – prepare a RAM disk driver for use (optional)

**SYNOPSIS** **STATUS ramDrv (void)**

**DESCRIPTION** This routine performs no real function, except to provide compatibility with earlier versions of **ramDrv** and to parallel the initialization function found in true disk device drivers. It also is used in **usrConfig.c** to link in the RAM disk driver when building VxWorks. Otherwise, there is no need to call this routine before using the RAM disk driver.

**RETURNS** OK, always.

**SEE ALSO** **ramDrv**

## rand()

|                      |                                                                                                                                                    |
|----------------------|----------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>          | <b>rand()</b> – generate a pseudo-random integer between 0 and <b>RAND_MAX</b> (ANSI)                                                              |
| <b>SYNOPSIS</b>      | <pre>int rand (void)</pre>                                                                                                                         |
| <b>DESCRIPTION</b>   | This routine generates a pseudo-random integer between 0 and <b>RAND_MAX</b> . The seed value for <b>rand()</b> can be reset with <b>srand()</b> . |
| <b>INCLUDE FILES</b> | <b>stdlib.h</b>                                                                                                                                    |
| <b>RETURNS</b>       | A pseudo-random integer.                                                                                                                           |
| <b>SEE ALSO</b>      | <b>ansiStdlib</b> , <b>srand()</b>                                                                                                                 |

---

## rawFsDevInit()

|                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
|--------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>        | <b>rawFsDevInit()</b> – associate a block device with raw volume functions                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| <b>SYNOPSIS</b>    | <pre>RAW_VOL_DESC *rawFsDevInit<br/>(<br/>    char *    pVolName,        /* volume name to be used with iosDevAdd */<br/>    BLK_DEV * pDevice         /* a pointer to a BLK_DEV or a CBIO_DEV_ID */<br/>)</pre>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
| <b>DESCRIPTION</b> | <p>This routine takes a block device created by a device driver and defines it as a raw file system volume. As a result, when high-level I/O operations, such as <b>open()</b> and <b>write()</b>, are performed on the device, the calls will be routed through <b>rawFsLib</b>.</p> <p>This routine associates <i>pVolName</i> with a device and installs it in the VxWorks I/O System's device table. The driver number used when the device is added to the table is that which was assigned to the raw library during <b>rawFsInit()</b>. (The driver number is kept in the global variable <b>rawFsDrvNum</b>.)</p> <p>The <i>pDevice</i> is a <b>CBIO_DEV_ID</b> or <b>BLK_DEV</b> ptr and contains configuration data describing the device and the addresses of routines which will be called to access device. These routines will not be called until they are required by subsequent I/O operations.</p> |
| <b>RETURNS</b>     | A pointer to the volume descriptor ( <b>RAW_VOL_DESC</b> ), or <b>NULL</b> if there is an error.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
| <b>SEE ALSO</b>    | <b>rawFsLib</b>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |

---

## rawFsInit()

**NAME** `rawFsInit()` – prepare to use the raw volume library

**SYNOPSIS**

```
STATUS rawFsInit
(
    int maxFiles          /* max no. of simultaneously open files */
)
```

**DESCRIPTION** This routine initializes the raw volume library. It must be called exactly once, before any other routine in the library. The argument specifies the number of file descriptors that may be open at once. This routine allocates and sets up the necessary memory structures and initializes semaphores.

This routine also installs raw volume library routines in the VxWorks I/O system driver table. The driver number assigned to `rawFsLib` is placed in the global variable `rawFsDrvNum`. This number will later be associated with system file descriptors opened to rawFs devices.

To enable this initialization, define `INCLUDE_RAWFS` in `configAll.h`; `rawFsInit()` will then be called from the root task, `usrRoot()`, in `usrConfig.c`.

**RETURNS** OK or ERROR.

**SEE ALSO** `rawFsLib`

---

## rawFsModeChange()

**NAME** `rawFsModeChange()` – modify the mode of a raw device volume

**SYNOPSIS**

```
void rawFsModeChange
(
    RAW_VOL_DESC * pVd,      /* pointer to volume descriptor */
    int          newMode     /* O_RDONLY/O_WRONLY/O_RDWR (both) */
)
```

**DESCRIPTION** This routine sets the device's mode to *newMode* by setting the mode field in the device structure. This routine should be called whenever the read and write capabilities are determined, usually after a ready change.

The driver's device initialization routine should initially set the mode to `O_RDWR` (*i.e.*, both `O_RDONLY` and `O_WRONLY`).

**RETURNS** N/A

**SEE ALSO** `rawFsLib`, `rawFsReadyChange()`

---

## rawFsReadyChange()

**NAME** `rawFsReadyChange()` – notify `rawFsLib` of a change in ready status

**SYNOPSIS**

```
void rawFsReadyChange
(
    RAW_VOL_DESC * pVd          /* pointer to volume descriptor */
)
```

**DESCRIPTION** This routine sets the volume descriptor state to `RAW_VD_READY_CHANGED`. It should be called whenever a driver senses that a device has come on-line or gone off-line, (*e.g.*, a disk has been inserted or removed).

After this routine has been called, the next attempt to use the volume will result in an attempted remount.

**RETURNS** N/A

**SEE ALSO** `rawFsLib`

---

## rawFsVolUnmount()

**NAME** `rawFsVolUnmount()` – disable a raw device volume

**SYNOPSIS**

```
STATUS rawFsVolUnmount
(
    RAW_VOL_DESC * pVd          /* pointer to volume descriptor */
)
```

**DESCRIPTION** This routine is called when I/O operations on a volume are to be discontinued. This is commonly done before changing removable disks. All buffered data for the volume is

written to the device (if possible), any open file descriptors are marked as obsolete, and the volume is marked as not mounted.

Because this routine will flush data from memory to the physical device, it should not be used in situations where the disk-change is not recognized until after a new disk has been inserted. In these circumstances, use the ready-change mechanism. (See the manual entry for `rawFsReadyChange()`.)

This routine may also be called by issuing an `ioctl()` call using the `FIOUNMOUNT` function code.

**RETURNS** OK, or **ERROR** if the routine cannot access the volume.

**SEE ALSO** `rawFsLib`, `rawFsReadyChange()`

---

## **rcmd()**

**NAME** `rcmd()` – execute a shell command on a remote machine

**SYNOPSIS**

```
int rcmd
(
    char * host,           /* host name or inet address */
    int  remotePort,     /* remote port to connect to (rshd) */
    char * localUser,     /* local user name */
    char * remoteUser,   /* remote user name */
    char * cmd,          /* command */
    int * fd2p,         /* if this pointer is non-zero, stderr */
                      /* socket is opened and socket descriptor is */
                      /* filled in */
)
```

**DESCRIPTION** This routine uses a remote shell daemon, `rshd`, to execute a command on a remote system. It is analogous to the BSD `rcmd()` routine.

Internally, this `rcmd()` implementation uses a `select()` call to wait for a response from the `rshd` daemon. If `rcmd()` receives a response within its timeout, `rcmd()` calls `accept()` and completes by returning a socket descriptor for the data generated on the remote machine.

The default timeout lets the `rcmd()` call wait forever. However, you can change the timeout value using the `RSH_STDERR_SETUP_TIMEOUT` parameter associated with the `NETWRS_REMLIB` configuration component.

- RETURNS** A socket descriptor if the remote shell daemon accepts, or **ERROR** if the remote command fails.
- ERRNO** **S\_remLib\_RSH\_ERROR**, **S\_remLib\_RSH\_STDERR\_SETUP\_FAILED**
- SEE ALSO** **remLib**, BSD reference entry for **rcmd()**

---

## rcvEtherAddrAdd()

- NAME** **rcvEtherAddrAdd()** – add a physical address into the linked list
- SYNOPSIS**
- ```
STATUS rcvEtherAddrAdd  
(  
    M2_IFINDEX *    pIfIndexEntry, /* the avl node */  
    unsigned char * pEnetAddr      /* the addr to be added */  
)
```
- DESCRIPTION** This function is a helper function for **rcvEtherAddrGet()**. It is called to add a single physical address into the linked list of addresses maintained by the AVL node.
- RETURNS** **OK**, if successful; **ERROR**, otherwise.
- SEE ALSO** **m2IfLib**

---

## rcvEtherAddrGet()

- NAME** **rcvEtherAddrGet()** – populate the **rcvAddr** fields for the **ifRcvAddressTable**
- SYNOPSIS**
- ```
STATUS rcvEtherAddrGet  
(  
    struct ifnet * pIfNet,          /* pointer to the interface's ifnet */  
    M2_IFINDEX *    pIfIndexEntry /* avl node */  
)
```
- DESCRIPTION** This function needs to be called to add all physical addresses for which an interface may receive or send packets. This includes unicast and multicast addresses. The address is inserted into the linked list maintained in the AVL node corresponding to the interface.

Given the `ifnet` struct and the AVL node corresponding to the interface, this function goes through all the physical addresses associated with this interface and adds them into the linked list.

**RETURNS** OK, if successful; **ERROR**, otherwise.

**SEE ALSO** `m2IfLib`

---

## rdCtl()

**NAME** `rdCtl()` – implement the ICMP router discovery control function

**SYNOPSIS**

```
STATUS rdCtl
(
    char * ifName,
    int   cmd,
    void* value           /* my be an int (set-cmds) or an int* */
                          /* (get-cmds) */
)
```

**DESCRIPTION** This routine allows a user to get and set router discovery parameters, and control the mode of operation.

**OPTIONS** The following section discuss the various flags that may be passed to `rdCtl()`.

### SET\_MODE

Set debug mode or exit router discovery

This flag does not require an *interface* to be specified it is best to specify `NULL`.

This flag is used in conjunction with the following values:

### MODE\_DEBUG\_ON

Turn debugging messages on.

```
rdCtl (NULL, SET_MODE, MODE_DEBUG_ON);
```

### MODE\_DEBUG\_OFF

Turn debugging messages off.

```
rdCtl (NULL, SET_MODE, MODE_DEBUG_OFF);
```

### MODE\_STOP

Exit from router discovery.

```
rdCtl (NULL, SET_MODE, MODE_STOP);
```

**rdCtl()**

**SET\_MIN\_ADVERT\_INT**

Set minimum advertisement interval in seconds

Specify the minimum time between advertisements in seconds. The minimum value allowed is 4 seconds, the maximum is 1800.

```
rdCtl (NULL, SET_MIN_ADVERT_INT, <seconds>);
```

**SET\_MAX\_ADVERT\_INT**

Set maximum advertisement interval in seconds

Specify the maximum time between advertisements in seconds. The minimum value allowed is 4 seconds, the maximum is 1800.

```
rdCtl (NULL, SET_MAX_ADVERT_INT, <seconds>);
```

**SET\_FLAG**

Set whether advertisements are sent on an interface.

If this flag is 1 then advertisements are sent on this interface. If it is 0 then they are not.

```
rdCtl (<interface>, SET_FLAG, <0 or 1>);
```

**SET\_ADVERT\_ADDRESS**

Set the IP address to which advertisements are sent.

Set the multicast IP address to which advertisements are sent.

```
rdCtl (<interface>, SET_ADVERT_ADDRESS, <multicast address>);
```

**SET\_ADVERT\_LIFETIME**

Set the lifetime for advertisements in seconds.

Set the lifetime in seconds to be contained in each advertisement.

```
rdCtl (<interface>, SET_ADVERT_LIFETIME, seconds);
```

**SET\_ADVERT\_PREF**

Set the preference level contained in advertisements.

```
rdCtl (<interface>, SET_ADVERT_PREF, value);
```

**GET\_MIN\_ADVERT\_INT**

Get the minimum advertisement interval.

```
rdCtl (NULL, GET_MIN_ADVERT_INT, &value);
```

**GET\_MAX\_ADVERT\_INT**

Get the maximum advertisement interval.

```
rdCtl (NULL, GET_MAX_ADVERT_INT, &value);
```

**GET\_FLAG**

Get the flag on an interface.

```
rdCtl (<interface>, GET_FLAG, &value);
```

**GET\_ADVERT\_ADDRESS**

Get the advertisement address for an interface.

```
rdCtl (<interface>, GET_ADVERT_ADDRESS, &value);
```

**GET\_ADVERT\_LIFETIME**

Get the advertisement lifetime.

```
rdCtl (<interface>, GET_ADVERT_LIFETIME, &value);
```

**GET\_ADVERT\_PREF**

Get the advertisement preference.

```
rdCtl (<interface>, GET_ADVERT_PREF, value);
```

**RETURNS** OK on success, **ERROR** on failure

**SEE ALSO** **rdiscLib**

---

## **rdisc()**

**NAME** **rdisc()** – implement the ICMP router discovery function

**SYNOPSIS** **void rdisc ()**

**DESCRIPTION** This routine is the entry point for the router discovery function. It allocates and initializes resources, listens for solicitation messages on the **ALL\_ROUTERS** (224.0.0.1) multicast address and processes the messages.

This routine usually runs until explicitly killed by a system operator, but can also be terminated cleanly (see **rdCtl()** routine).

**RETURNS** N/A

**SEE ALSO** **rdiscLib**

## **rdiscIfReset()**

|                    |                                                                                                                                                                                                                                                                                                      |
|--------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>        | <b>rdiscIfReset()</b> – check for new or removed interfaces for router discovery                                                                                                                                                                                                                     |
| <b>SYNOPSIS</b>    | <b>STATUS</b> <b>rdiscIfReset</b> ()                                                                                                                                                                                                                                                                 |
| <b>DESCRIPTION</b> | This routine <i>must</i> be called any time an interface is added to or removed from the system so that the router discovery code can deal with this case. Failure to do so will cause the sending of packets on missing interfaces to fail as well as no transmission of packets on new interfaces. |
| <b>SEE ALSO</b>    | <b>rdiscLib</b>                                                                                                                                                                                                                                                                                      |

---

## **rdiscInit()**

|                    |                                                                                                                                                          |
|--------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>        | <b>rdiscInit()</b> – initialize the ICMP router discovery function                                                                                       |
| <b>SYNOPSIS</b>    | <b>STATUS</b> <b>rdiscInit</b> ()                                                                                                                        |
| <b>DESCRIPTION</b> | This routine allocates resources for the router discovery function. Since it called in the <b>rdisc()</b> routine, it should not be called subsequently. |
| <b>RETURNS</b>     | <b>OK</b> on successful initialization, <b>ERROR</b> otherwise                                                                                           |
| <b>SEE ALSO</b>    | <b>rdiscLib</b>                                                                                                                                          |

---

## **rdiscLibInit()**

**NAME** `rdiscLibInit()` – initialize router discovery

**SYNOPSIS**

```
void rdiscLibInit
(
    int priority,          /* Priority of router discovery task. */
    int options,          /* Options to taskSpawn(1) for router */
                          /* discovery task. */
    int stackSize         /* Stack size for router discovery task. */
)
```

**DESCRIPTION** This routine links the ICMP Router Discovery facility into the VxWorks system. The arguments are the task's priority, options and stack size.

**RETURNS** N/A

**SEE ALSO** `rdiscLib`

---

## **rdiscTimerEvent()**

**NAME** `rdiscTimerEvent()` – called after watchdog timeout

**SYNOPSIS**

```
void rdiscTimerEventRestart
(
    int stackNum
)
```

**DESCRIPTION** This routine is called when a new advertisement is to be sent.

**RETURNS** N/A

**SEE ALSO** `rdiscLib`

**read()**

---

**read()****NAME** read() – read bytes from a file or device

**SYNOPSIS**

```
int read
(
    int    fd,                /* file descriptor from which to read */
    char * buffer,           /* pointer to buffer to receive bytes */
    size_t maxbytes         /* max no. of bytes to read into buffer */
)
```

**DESCRIPTION** This routine reads a number of bytes (less than or equal to *maxbytes*) from a specified file descriptor and places them in *buffer*. It calls the device driver to do the work.**RETURNS** The number of bytes read (between 1 and *maxbytes*, 0 if end of file), or **ERROR** if the file descriptor does not exist, the driver does not have a read routine, or the driver returns **ERROR**. If the driver does not have a read routine, **errno** is set to **ENOTSUP**.**SEE ALSO** ioLib

---

**readdir()****NAME** readdir() – read one entry from a directory (POSIX)

**SYNOPSIS**

```
struct dirent *readdir
(
    DIR * pDir                /* pointer to directory descriptor */
)
```

**DESCRIPTION** This routine obtains directory entry data for the next file from an open directory. The *pDir* parameter is the pointer to a directory descriptor (DIR) which was returned by a previous **opendir()**.

This routine returns a pointer to a **dirent** structure which contains the name of the next file. Empty directory entries and MS-DOS volume label entries are not reported. The name of the file (or subdirectory) described by the directory entry is returned in the **d\_name** field of the **dirent** structure. The name is a single null-terminated string.

The returned **dirent** pointer will be **NULL**, if it is at the end of the directory or if an error occurred. Because there are two conditions which might cause **NULL** to be returned, the

task's error number (**errno**) must be used to determine if there was an actual error. Before calling **readdir()**, set **errno** to **OK**. If a **NULL** pointer is returned, check the new value of **errno**. If **errno** is still **OK**, the end of the directory was reached; if not, **errno** contains the error code for an actual error which occurred.

**RETURNS** A pointer to a **dirent** structure, or **NULL** if there is an end-of-directory marker or error.

**SEE ALSO** **dirLib**, **opendir()**, **closedir()**, **rewinddir()**, **ls()**

---

## realloc()

**NAME** **realloc()** – reallocate a block of memory (ANSI)

**SYNOPSIS**

```
void *realloc
(
    void * pBlock,          /* block to reallocate */
    size_t newSize         /* new block size */
)
```

**DESCRIPTION** This routine changes the size of a specified block of memory and returns a pointer to the new block of memory. The contents that fit inside the new size (or old size if smaller) remain unchanged. The memory alignment of the new block is not guaranteed to be the same as the original block.

**RETURNS** A pointer to the new block of memory, or **NULL** if the call fails.

**SEE ALSO** **memLib**, *American National Standard for Information Systems -Programming Language - C, ANSI X3.159-1989: General Utilities (**stdlib.h**)*

## reboot()

**NAME** `reboot()` – reset network devices and transfer control to boot ROMs

**SYNOPSIS**

```
void reboot
(
    int startType          /* how the boot ROMs will reboot */
)
```

**DESCRIPTION** This routine returns control to the boot ROMs after calling a series of preliminary shutdown routines that have been added via `rebootHookAdd()`, including routines to reset all network devices. After calling the shutdown routines, interrupts are locked, all caches are cleared, and control is transferred to the boot ROMs.

The bit values for *startType* are defined in `sysLib.h`:

**BOOT\_NORMAL** (0x00)

causes the system to go through the countdown sequence and try to reboot VxWorks automatically. Memory is not cleared.

**BOOT\_NO\_AUTOBOOT** (0x01)

causes the system to display the VxWorks boot prompt and wait for user input to the boot ROM monitor. Memory is not cleared.

**BOOT\_CLEAR** (0x02)

the same as **BOOT\_NORMAL**, except that memory is cleared.

**BOOT\_QUICK\_AUTOBOOT** (0x04)

the same as **BOOT\_NORMAL**, except the countdown is shorter.

**RETURNS** N/A

**SEE ALSO** `rebootLib`, `sysToMonitor()`, `rebootHookAdd()`, *VxWorks Programmer's Guide: Target Shell*, *windsh*, *Tornado User's Guide: Shell*

---

## rebootHookAdd()

**NAME** rebootHookAdd() – add a routine to be called at reboot

**SYNOPSIS**

```
STATUS rebootHookAdd
(
    FUNCPTR rebootHook      /* routine to be called at reboot */
)
```

**DESCRIPTION** This routine adds the specified routine to a list of routines to be called when VxWorks is rebooted. The specified routine should be declared as follows:

```
void rebootHook
(
    int startType /* startType is passed to all hooks */
)
```

**RETURNS** OK, or ERROR if memory is insufficient.

**SEE ALSO** rebootLib, reboot()

---

## recv()

**NAME** recv() – receive data from a socket

**SYNOPSIS**

```
int recv
(
    int s,                /* socket to receive data from */
    char * buf,           /* buffer to write data to */
    int bufLen,          /* length of buffer */
    int flags              /* flags to underlying protocols */
)
```

**DESCRIPTION** This routine receives data from a connection-based (stream) socket.

The maximum length of *buf* is subject to the limits on TCP buffer size; see the discussion of `SO_RCVBUF` in the `setsockopt()` manual entry.

You may OR the following values into the *flags* parameter with this operation:

**MSG\_OOB** (0x1)

Out-of-band data.

**MSG\_PEEK** (0x2)

Return data without removing it from socket.

**RETURNS** The number of bytes received, or **ERROR** if the call fails.

**SEE ALSO** **sockLib**, **setsockopt()**

---

## recvfrom()

**NAME** **recvfrom()** – receive a message from a socket

**SYNOPSIS**

```
int recvfrom
(
    int          s,          /* socket to receive from */
    char *       buf,       /* pointer to data buffer */
    int          buflen,    /* length of buffer */
    int          flags,     /* flags to underlying protocols */
    struct sockaddr * from, /* where to copy sender's addr */
    int *        pFromLen /* value/result length of from */
)
```

**DESCRIPTION**

This routine receives a message from a datagram socket regardless of whether it is connected. If *from* is non-zero, the address of the sender's socket is copied to it. The value-result parameter *pFromLen* should be initialized to the size of the *from* buffer. On return, *pFromLen* contains the actual size of the address stored in *from*.

The maximum length of *buf* is subject to the limits on UDP buffer size; see the discussion of **SO\_RCVBUF** in the **setsockopt()** manual entry.

You may OR the following values into the *flags* parameter with this operation:

**MSG\_OOB** (0x1)

Out-of-band data.

**MSG\_PEEK** (0x2)

Return data without removing it from socket.

**RETURNS** The number of number of bytes received, or **ERROR** if the call fails.

**SEE ALSO** **sockLib**, **setsockopt()**

---

## recvmsg()

**NAME** recvmsg() – receive a message from a socket

**SYNOPSIS**

```
int recvmsg
(
    int          sd,          /* socket to receive from */
    struct msghdr * mp,      /* scatter-gather message header */
    int          flags       /* flags to underlying protocols */
)
```

**DESCRIPTION** This routine receives a message from a datagram socket. It may be used in place of **recvfrom()** to decrease the overhead of breaking down the message-header structure **msghdr** for each message.

For BSD 4.4 sockets a copy of the *mp*>msg\_iov array will be made. This requires a cluster from the network stack system pool of size *mp*>msg\_iovlen \* sizeof (struct iovec) or 8 bytes.

**RETURNS** The number of bytes received, or **ERROR** if the call fails.

**SEE ALSO** sockLib

---

## rld()

**NAME** rld() – reload an object module

**SYNOPSIS**

```
MODULE_ID rld
(
    void * nameOrId,        /* name or ID of the object module file */
    int   options          /* options used for unloading */
)
```

**DESCRIPTION** This routine unloads a specified object module from the system, and then calls **loadModule()** to load a new copy of the same name.

If the file was originally loaded using a complete pathname, then **rld()** will use the complete name to locate the file. If the file was originally loaded using a partial pathname, then the current working directory must be changed to the working directory in use at the time of the original load.

**remCurIdGet()**

Valid values for the options parameter are the same as those allowed for the function **unld()**.

This routine is a **shell command**. That is, it is designed to be used only in the shell, and not in code running on the target. In future releases, calling **reld()** directly from code may not be supported.

**RETURNS** A module ID (type **MODULE\_ID**), or **NULL**.

**SEE ALSO** **unldLib**, **unld()**

---

## remCurIdGet()

**NAME** **remCurIdGet()** – get the current user name and password

**SYNOPSIS**

```
void remCurIdGet
(
    char * user,           /* where to return current user name */
    char * passwd        /* where to return current password */
)
```

**DESCRIPTION** This routine gets the user name and password currently used for remote host access privileges and copies them to *user* and *passwd*. Either parameter can be initialized to **NULL**, and the corresponding item will not be passed.

**RETURNS** N/A

**SEE ALSO** **remLib**, **iam()**, **whoami()**

---

## remCurIdSet()

**NAME** **remCurIdSet()** – set the remote user name and password

**SYNOPSIS**

```
STATUS remCurIdSet
(
    char * newUser,       /* user name to use on remote */
    char * newPasswd     /* password to use on remote (NULL = none) */
)
```

|                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
|--------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>DESCRIPTION</b> | <p>This routine specifies the user name that will have access privileges on the remote machine. The user name must exist in the remote machine's <code>/etc/passwd</code>, and if it has been assigned a password, the password must be specified in <code>newPasswd</code>.</p> <p>Either parameter can be <code>NULL</code>, and the corresponding item will not be set.</p> <p>The maximum length of the user name and the password is <code>MAX_IDENTITY_LEN</code>(defined in <code>remLib.h</code>).</p> <hr/> <p><b>NOTE:</b> A more convenient version of this routine is <code>iam()</code>, which is intended to be used from the shell.</p> <hr/> |
| <b>RETURNS</b>     | <code>OK</code> , or <code>ERROR</code> if the name or password is too long.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| <b>SEE ALSO</b>    | <code>remLib</code> , <code>iam()</code> , <code>whoami()</code>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |

---

## remove()

|                    |                                                                                                                                                                                               |
|--------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>        | <code>remove()</code> – remove a file (ANSI)                                                                                                                                                  |
| <b>SYNOPSIS</b>    | <pre><b>STATUS</b> remove (     const char * name          /* name of the file to remove */ )</pre>                                                                                           |
| <b>DESCRIPTION</b> | <p>This routine deletes a specified file. It calls the driver for the particular device on which the file is located to do the work.</p>                                                      |
| <b>RETURNS</b>     | <code>OK</code> if there is no delete routine for the device or the driver returns <code>OK</code> ; <code>ERROR</code> if there is no such device or the driver returns <code>ERROR</code> . |
| <b>SEE ALSO</b>    | <code>ioLib</code> , <i>American National Standard for Information Systems -Programming Language - C, ANSI X3.159-1989: Input/Output (<code>stdio.h</code>)</i>                               |

---

## rename()

**NAME** rename() – change the name of a file

**SYNOPSIS**

```
int rename
(
    const char * oldname,      /* name of file to rename */
    const char * newname      /* name with which to rename file */
)
```

**DESCRIPTION** This routine changes the name of a file from *oldfile* to *newfile*.

---

**NOTE:** Only certain devices support **rename()**. To confirm that your device supports it, consult the respective **xxDrv** or **xxFs** listings to verify that ioctl **FIORENAME** exists. For example, **dosFs** and **rt11Fs** support **rename()**, but **netDrv** and **nfsDrv** do not.

---

**RETURNS** OK, or ERROR if the file could not be opened or renamed.

**SEE ALSO** ioLib

---

## repeat()

**NAME** repeat() – spawn a task to call a function repeatedly

**SYNOPSIS**

```
int repeat
(
    int      n,                /* no. of times to call func (0=forever) */
    FUNCPTR func,             /* function to call repeatedly */
    int      arg1,             /* first of eight args to pass to func */
    int      arg2,
    int      arg3,
    int      arg4,
    int      arg5,
    int      arg6,
    int      arg7,
    int      arg8
)
```

**DESCRIPTION** This command spawns a task that calls a specified function *n* times, with up to eight of its arguments. If *n* is 0, the routine is called endlessly, or until the spawned task is deleted.

---

**NOTE:** The task is spawned using **sp()**. See the description of **sp()** for details about priority, options, stack size, and task ID.

---

**RETURNS** A task ID, or **ERROR** if the task cannot be spawned.

**SEE ALSO** **usrLib, repeatRun(), sp()**, *VxWorks Programmer's Guide: Target Shell*, **windsh**, *Tornado User's Guide: Shell*

---

## repeatRun()

**NAME** **repeatRun()** – call a function repeatedly

**SYNOPSIS**

```
void repeatRun
(
    int      n,                /* no. of times to call func (0=forever) */
    FUNCPTR func,            /* function to call repeatedly */
    int      arg1,           /* first of eight args to pass to func */
    int      arg2,
    int      arg3,
    int      arg4,
    int      arg5,
    int      arg6,
    int      arg7,
    int      arg8
)
```

**DESCRIPTION** This command calls a specified function *n* times, with up to eight of its arguments. If *n* is 0, the routine is called endlessly.

Normally, this routine is called only by **repeat()**, which spawns it as a task.

**RETURNS** N/A

**SEE ALSO** **usrLib, repeat()**, *VxWorks Programmer's Guide: Target Shell*

---

## resolvDNComp()

**NAME** `resolvDNComp()` – compress a DNS name in a DNS packet

**SYNOPSIS**

```
int resolvDNComp
(
    const u_char * exp_dn,    /* ptr to the expanded domain name */
    u_char *      comp_dn,    /* ptr to where to output the compressed name */
    int           length,     /* length of the buffer pointed by comp_dn */
    u_char * *    dnptrs,     /* ptr to a ptr list of compressed names */
    u_char * *    lastdnptr /* ptr to the last entry pointed by dnptrs */
)
```

**DESCRIPTION** This routine takes the expanded domain name referenced in the *exp\_dn* parameter, compresses it, and stores the compressed name in the location pointed to by the *comp\_dn* parameter. The *length* parameter passes in the length of the buffer starting at *comp\_dn*. The *dnptrs* parameter is a pointer to a list of pointers to previously compressed names. The *lastdnptr* parameter points to the last entry in the *dnptrs* array.

**RETURNS** The size of the compressed name, or `ERROR`.

**SEE ALSO** `resolvLib`, `resolvGetHostByName()`, `resolvGetHostByAddr()`, `resolvDNExpand()`, `resolvInit()`, `resolvSend()`, `resolvParamsSet()`, `resolvParamsGet()`, `resolvMkQuery()`, `resolvQuery()`

---

## resolvDNExpand()

**NAME** `resolvDNExpand()` – expand a DNS compressed name from a DNS packet

**SYNOPSIS**

```
int resolvDNExpand
(
    const u_char * msg,       /* ptr to the start of the DNS packet */
    const u_char * eomorig,   /* ptr to the last location +1 of the DNS */
                             /* packet */
    const u_char * comp_dn,   /* ptr to the compressed domain name */
    u_char *      exp_dn,     /* ptr to where the expanded DN is output */
    int           length      /* length of the buffer pointed by expd_dn */
)
```

- DESCRIPTION** This functions expands a compressed DNS name from a DNS packet. The *msg* parameter points to that start of the DNS packet. The *comorig* parameter points to the last location of the DNS packet plus 1. The *comp\_dn* parameter points to the compress domain name, and *exp\_dn* parameter expects a pointer to a buffer. Upon function completion, this buffer contains the expanded domain name. Use the *length* parameter to pass in the size of the buffer referenced by the *exp\_dn* parameter.
- RETURNS** The length of the expanded domain name, or **ERROR** on failure.
- SEE ALSO** `resolvLib`, `resolvGetHostByName()`, `resolvGetHostByAddr()`, `resolvInit()`, `resolvDNComp()`, `resolvSend()`, `resolvParamsSet()`, `resolvParamsGet()`, `resolvMkQuery()`, `resolvQuery()`

---

## resolvGetHostByAddr()

- NAME** `resolvGetHostByAddr()` – query the DNS server for the host name of an IP address
- SYNOPSIS**

```
struct hostent * resolvGetHostByAddr
(
    const char * pInetAddr,
    char *      pHostBuf,
    int        bufLen
)
```

- DESCRIPTION** This function returns a **hostent** structure, which is defined as follows:

```
struct hostent
{
    char * h_name;           /* official name of host */
    char ** h_aliases;      /* alias list */
    int h_addrtype;         /* address type */
    int h_length;           /* length of address */
    char ** h_addr_list;    /* list of addresses from name server */
    unsigned int h_ttl;     /* Time to Live in Seconds for this entry */
}
```

The **h\_aliases** and **h\_addr\_list** vectors are NULL-terminated. For a locally resolved entry **h\_ttl** is always 60 (an externally resolved entry may also have a TTL of 60 depending on its age but it is usually much higher).

The *pinetAddr* parameter passes in the IP address (in network byte order) for the host whose name you want to discover. The *pBuf* and *bufLen* parameters specify the location

and size (512 bytes or more) of the buffer that is to receive the **hostent** structure.  
**resolvGetHostByAddr()** returns host addresses are returned in network byte order.

**RETURNS** A pointer to a **hostent** structure if the host is found, or **NULL** if the parameters are invalid, host is not found, or the buffer is too small.

**ERRNO**

- S\_resolvLib\_INVALID\_PARAMETER
- S\_resolvLib\_BUFFER\_2\_SMALL
- S\_resolvLib\_TRY\_AGAIN
- S\_resolvLib\_HOST\_NOT\_FOUND
- S\_resolvLib\_NO\_DATA
- S\_resolvLib\_NO\_RECOVERY

**SEE ALSO** **resolvLib**, **resolvGetHostByName()**, **resolvInit()**, **resolvDNExpand()**, **resolvDNComp()**, **resolvSend()**, **resolvParamsSet()**, **resolvParamsGet()**, **resolvMkQuery()**, **resolvQuery()**

---

## resolvGetHostByName()

**NAME** **resolvGetHostByName()** – query the DNS server for the IP address of a host

**SYNOPSIS**

```
struct hostent * resolvGetHostByName
(
    char * pHostName, /* ptr to the name of the host */
    char * pHostBuf, /* ptr to the buffer used by hostent structure */
    int  bufLen      /* length of the buffer */
)
```

**DESCRIPTION** This function returns a **hostent** structure. This structure is defined as follows:

```
struct  hostent
{
    char *  h_name;          /* official name of host */
    char ** h_aliases;      /* alias list */
    int     h_addrtype;     /* address type */
    int     h_length;       /* length of address */
    char ** h_addr_list;    /* list of addresses from name server */
    unsigned int h_ttl;     /* Time to Live in Seconds for this entry */
}
```

The **h\_aliases** and **h\_addr\_list** vectors are **NULL**-terminated. For a locally resolved entry **h\_ttl** is always 60 (an externally resolved entry may also have a TTL of 60 depending on its age but it is usually much higher).

Specify the host you want to query in *pHostname*. Use *pBuf* and *bufLen* to specify the location and size of a buffer to receive the **hostent** structure and its associated contents. Host addresses are returned in network byte order. Given the information this routine retrieves, the *pBuf* buffer should be 512 bytes or larger.

- RETURNS** A pointer to a **hostent** structure if the host is found, or **NULL** if the parameters are invalid, the host is not found, or the buffer is too small.
- ERRNO** **S\_resolvLib\_INVALID\_PARAMETER**  
**S\_resolvLib\_BUFFER\_2\_SMALL**  
**S\_resolvLib\_TRY\_AGAIN**  
**S\_resolvLib\_HOST\_NOT\_FOUND**  
**S\_resolvLib\_NO\_DATA**  
**S\_resolvLib\_NO\_RECOVERY**
- SEE ALSO** **resolvLib**, **resolvInit()**, **resolvGetHostByAddr()**, **resolvDNExpand()**, **resolvDNComp()**, **resolvSend()**, **resolvParamsSet()**, **resolvParamsGet()**, **resolvMkQuery()**, **resolvQuery()**

---

## resolvInit()

- NAME** **resolvInit()** – initialize the resolver library
- SYNOPSIS**
- ```
STATUS resolvInit
(
    char * pNameServer,          /* pointer to Name server IP address */
    char * pDefaultDomainName, /* default domain name */
    FUNCPTR pdnsDebugRtn       /* function ptr to debug routine */
)
```
- DESCRIPTION** This function initializes the resolver. *pNameServer* is a single IP address for a name server in dotted decimal notation. *pDefaultDomainName* is the default domain name to be appended to names without a dot. The function pointer *pdnsDebugRtn* is set to the resolver debug function. Additional name servers can be configured using the function **resolvParamsSet()**.
- RETURNS** **OK** or **ERROR**.
- SEE ALSO** **resolvLib**, **resolvGetHostByName()**, **resolvGetHostByAddr()**, **resolvDNExpand()**, **resolvDNComp()**, **resolvSend()**, **resolvParamsSet()**, **resolvParamsGet()**, **resolvQuery()**

## resolvMkQuery()

**NAME** resolvMkQuery() – create all types of DNS queries

**SYNOPSIS**

```
int resolvMkQuery
(
    int          op,          /* set to desire query QUERY or IQUERY */
    const char * dname,      /* domain name to be use in the query */
    int          class,      /* query class for IP is C_IN */
    int          type,       /* type is T_A, T_PTR, ... */
    const char * data,       /* resource Record (RR) data */
    int          datalen,    /* length of the RR */
    const char * newrr_in,   /* not used always set to NULL */
    char *       buf,        /* out of the constructed query */
    int          buflen      /* length of the buffer for the query */
)
```

**DESCRIPTION** This routine uses the input parameters to create a domain name query. You can set the *op* parameter to *QUERY* or *IQUERY*. Specify the domain name in *dname*, the class in *class*, the query type in *type*. Valid values for type include *T\_A*, *T\_PTR*, and so on. Use *data* to add Resource Record data to the query. Use *datalen* to pass in the length of the data buffer. Set *newrr\_in* to *NULL*. This parameter is reserved for future use. The *buf* parameter expects a pointer to the output buffer for the constructed query. Use *buflen* to pass in the length of the buffer referenced in *buf*.

**RETURNS** The length of the constructed query or *ERROR*.

**SEE ALSO** *resolvLib*, *resolvGetHostByName()*, *resolvGetHostByAddr()*, *resolvDNExpand()*, *resolvDNComp()*, *resolvSend()*, *resolvParamsSet()*, *resolvParamsGet()*, *resolvInit()*, *resolvQuery()*

---

## resolvParamsGet()

**NAME** resolvParamsGet() – get the parameters which control the resolver library

**SYNOPSIS**

```
void resolvParamsGet
(
    RESOLV_PARAMS_S * pResolvParams /* ptr to resolver parameter struct */
)
```

**DESCRIPTION** This routine copies the resolver parameters to the `RESOLV_PARAMS_S` structure referenced in the `pResolvParams` parameter. The `RESOLV_PARAMS_S` structure is defined in `resolvLib.h` as follows:

```
typedef struct
{
    char    queryOrder;
    char    domainName [MAXDNAME];
    char    nameServersAddr [MAXNS] [MAXIPADDRLLEN];
} RESOLV_PARAMS_S;
```

Typically, you call this function just before calling `resolvParamsSet()`. The `resolvParamsGet()` call populates the `RESOLV_PARAMS_S` structure. You can then modify the default values just before calling `resolvParamsSet()`.

**RETURNS** N/A

**SEE ALSO** `resolvLib`, `resolvGetHostByName()`, `resolvGetHostByAddr()`, `resolvDNExpand()`, `resolvDNComp()`, `resolvSend()`, `resolvParamsSet()`, `resolvInit()`, `resolvMkQuery()`, `resolvQuery()`

---

## resolvParamsSet()

**NAME** `resolvParamsSet()` – set the parameters which control the resolver library

**SYNOPSIS**

```
STATUS resolvParamsSet
(
    RESOLV_PARAMS_S * pResolvParams /* ptr to resolver parameter struct */
)
```

**DESCRIPTION** This routine sets the resolver parameters. `pResolvParams` passes in a pointer to a `RESOLV_PARAMS_S` structure, which is defined as follows:

```
typedef struct
{
    char    queryOrder;
    char    domainName [MAXDNAME];
    char    nameServersAddr [MAXNS] [MAXIPADDRLLEN];
} RESOLV_PARAMS_S;
```

Use the members of this structure to specify the settings you want to apply to the resolver. It is important to remember that multiple tasks can use the resolver library and that the settings specified in this `RESOLV_PARAMS_S` structure affect all queries from all tasks. In

addition, you should set resolver parameters at initialization and not while queries could be in progress. Otherwise, the results of the query are unpredictable.

Before calling **resolvParamsSet()**, you should first call **resolvParamsGet()** to populate a **RESOLV\_PARAMS\_S** structure with the current settings. Then you change the values of the members that interest you.

Valid values for the **queryOrder** member of **RESOLV\_PARAMS\_S** structure are defined in **resolvLib.h**. Set the **domainName** member to the domain to which this resolver belongs. Set the **nameServersAddr** member to the IP addresses of the DNS server that the resolver can query. You must specify the IP addresses in standard dotted decimal notation. This function tries to validate the values in the **queryOrder** and **nameServerAddr** members. This function does not try to validate the domain name.

**RETURNS** OK if the parameters are valid, **ERROR** otherwise.

**SEE ALSO** **resolvLib**, **resolvGetHostByName()**, **resolvGetHostByAddr()**, **resolvDNExpand()**, **resolvDNComp()**, **resolvSend()**, **resolvInit()**, **resolvParamsGet()**, **resolvMkQuery()**, **resolvQuery()**

---

## resolvQuery()

**NAME** **resolvQuery()** – construct a query, send it, wait for a response

**SYNOPSIS**

```
int resolvQuery
(
    char *   name,           /* domain name */
    int      class,         /* query class for IP is C_IN */
    int      type,          /* type is T_A, T_PTR, ... */
    u_char * answer,        /* buffer to put answer */
    int      anslen         /* length of answer buffer */
)
```

**DESCRIPTION** This routine constructs a query for the domain specified in the *name* parameter. The *class* parameter specifies the class of the query. The *type* parameter specifies the type of query. The routine then sends the query to the DNS server. When the server responds, the response is validated and copied to the buffer you supplied in the *answer* parameter. Use the *anslen* parameter to pass in the size of the buffer referenced in *answer*.

**RETURNS** The length of the response or **ERROR**.

<b>ERRNO</b>	S_resolvLib_TRY_AGAIN S_resolvLib_HOST_NOT_FOUND S_resolvLib_NO_DATA S_resolvLib_NO_RECOVERY
<b>SEE ALSO</b>	resolvLib, resolvGetHostByName(), resolvGetHostByAddr(), resolvDNExpand(), resolvDNComp(), resolvInit(), resolvParamsSet(), resolvParamsGet(), resolvMkQuery()

---

## resolvSend()

<b>NAME</b>	resolvSend() – send a pre-formatted query and return the answer
<b>SYNOPSIS</b>	<pre>int resolvSend (     const char * buf,          /* pre-formatted query */     int         buflen,       /* length of query */     char *      answer,       /* buffer for answer */     int         anslen        /* length of answer */ )</pre>
<b>DESCRIPTION</b>	This routine takes a pre-formatted DNS query and sends it to the domain server. Use <i>buf</i> to pass in a pointer to the query. Use <i>buflen</i> to pass in the size of the buffer referenced in <i>buf</i> . The <i>answer</i> parameter expects a pointer to a buffer into which this routine can write the answer retrieved from the server. Use <i>anslen</i> to pass in the size of the buffer you have provided in <i>anslen</i> .
<b>RETURNS</b>	The length of the response or <b>ERROR</b> .
<b>ERRNO</b>	S_resolvLib_TRY_AGAIN ECONNREFUSE ETIMEDOU
<b>SEE ALSO</b>	resolvLib, resolvGetHostByName(), resolvGetHostByAddr(), resolvDNExpand(), resolvDNComp(), resolvInit(), resolvParamsSet(), resolvParamsGet(), resolvMkQuery(), resolvQuery()

## rewind()

<b>NAME</b>	<b>rewind()</b> – set the file position indicator to the beginning of a file (ANSI)
<b>SYNOPSIS</b>	<pre>void rewind (     FILE * fp                /* stream */ )</pre>
<b>DESCRIPTION</b>	<p>This routine sets the file position indicator for a specified stream to the beginning of the file.</p> <p>It is equivalent to:</p> <pre>(void) fseek (fp, 0L, SEEK_SET);</pre> <p>except that the error indicator for the stream is cleared.</p>
<b>INCLUDE FILES</b>	stdio.h
<b>RETURNS</b>	N/A
<b>SEE ALSO</b>	ansiStdio, fseek(), ftell()

---

## rewinddir()

<b>NAME</b>	<b>rewinddir()</b> – reset position to the start of a directory (POSIX)
<b>SYNOPSIS</b>	<pre>void rewinddir (     DIR * pDir              /* pointer to directory descriptor */ )</pre>
<b>DESCRIPTION</b>	<p>This routine resets the position pointer in a directory descriptor (DIR). The <i>pDir</i> parameter is the directory descriptor pointer that was returned by <b>opendir()</b>.</p> <p>As a result, the next <b>readdir()</b> will cause the current directory data to be read in again, as if an <b>opendir()</b> had just been performed. Any changes in the directory that have occurred since the initial <b>opendir()</b> will now be visible. The first entry in the directory will be returned by the next <b>readdir()</b>.</p>

**RETURNS** N/A

**SEE ALSO** `dirLib`, `opendir()`, `readdir()`, `closedir()`

---

## rindex()

**NAME** `rindex()` – find the last occurrence of a character in a string

**SYNOPSIS**

```
char *rindex
(
    const char * s,          /* string in which to find character */
    int        c            /* character to find in string */
)
```

**DESCRIPTION** This routine finds the last occurrence of character *c* in string *s*.

**RETURNS** A pointer to *c*, or NULL if *c* is not found.

**SEE ALSO** `bLib`

---

## ripAddrXtract()

**NAME** `ripAddrXtract()` – extract socket address pointers from the route message

**SYNOPSIS**

```
void ripAddrXtract
(
    ROUTE_INFO *      pRtInfo,    /* Route information message */
    struct sockaddr * * pDstAddr, /* Where to store the Destination */
                                /* addr pointer */
    struct sockaddr * * pNetmask, /* Where to store the netmask pointer*/
    struct sockaddr * * pGateway, /* Where to store the Gateway addr */
                                /* pointer */
    struct sockaddr * * pOldGateway /* Where to store the Old gateway */
                                /* addr (if any) pointer */
)
```

<b>DESCRIPTION</b>	<p>This routine extracts the socket addresses from the route message in <i>pRtInfo</i> and uses the other parameters to return pointers to the extracted messages.</p> <p><i>pRtInfo</i> Passes in a pointer to a route information message.</p> <p><i>pDstAddr</i> Returns a pointer to the destination address.</p> <p><i>pNetmask</i> Returns a pointer to the netmask.</p> <p><i>pGateway</i> Returns a pointer to the gateway address.</p> <p><i>pOldGateway</i> Returns a pointer to the OLD gateway address if it exists.</p> <p>If the route message doesn't specify an address, the corresponding address pointer is set to NULL</p>
<b>RETURNS</b>	N/A
<b>ERRNO</b>	N/A
<b>SEE ALSO</b>	<b>ripLib</b>

---

## ripAuthHook()

<b>NAME</b>	<b>ripAuthHook()</b> – sample authentication hook
<b>SYNOPSIS</b>	<pre><b>STATUS</b> ripAuthHook (     char *    pKey,          /* rip2IfConfAuthKey entry from MIB-II family */     RIP_PKT * pRip         /* received RIP message */ )</pre>
<b>DESCRIPTION</b>	<p>This hook demonstrates one possible authentication mechanism. It rejects all RIP-2 messages that used simple password authentication since they did not match the key contained in the MIB variable. All other RIP-2 messages are also rejected since no other authentication type is supported and all RIP-1 messages are also rejected, as recommended by the RFC specification. This behavior is the same as if no hook were installed.</p>

**RETURNS** OK, if message is acceptable; or **ERROR** otherwise.

**SEE ALSO** `ripLib`

---

## ripAuthHookAdd()

**NAME** `ripAuthHookAdd()` – add an authentication hook to a RIP interface

**SYNOPSIS**

```
STATUS ripAuthHookAdd
(
    char*   pIpAddr,           /* IP address in dotted decimal notation */
    FUNCPTR pAuthHook         /* routine to handle message authentication */
)
```

**DESCRIPTION** This routine installs a hook routine to validate incoming RIP messages for a registered interface given by *pIpAddr*. (Interfaces created or changed after a RIP session has started may be installed/updated with the `ripIfSearch()` and `ripIfReset()` routines). The hook is only called if an SNMP agent enables authentication for the corresponding interface. It uses the following prototype:

```
STATUS ripAuthHookRtn (char *pKey, RIP_PKT *pRip);
```

The first argument contains the authentication key for the message stored in the `rip2IfConfAuthKey` MIB variable and the second argument uses the `RIP_PKT` structure (defined in `rip/ripLib.h`) to access the message body. The routine must return **OK** if the message is acceptable, or **ERROR** otherwise. All RIP-2 messages sent to that routine already contain an authentication entry, but have not been verified. (Any unauthenticated RIP-2 messages have already been discarded as required by the RFC specification). RIP-1 messages may be accepted or rejected. RIP-2 messages requesting simple password authentication that match the key are accepted automatically before the hook is called. The remaining RIP-2 messages either did not match that key or are using an unknown authentication type. If any messages are rejected, the MIB-II counters are updated appropriately outside of the hook routine.

The current RIP implementation contains a sample authentication hook that you may add as follows:

```
if (ripAuthHookAdd ("90.0.0.1", ripAuthHook) == ERROR)
    logMsg ("Unable to add authorization hook.\n", 0, 0, 0, 0, 0, 0);
```

The sample routine supports only simple password authentication against the key included in the MIB variable. Since all such messages have already been accepted, all RIP-2 messages received by the routine are discarded. All RIP-1 messages are also discarded, so the hook actually has no effect. The body of that routine is:

```
STATUS ripAuthHook
(
    char *      pKey,      /* rip2IfConfAuthKey entry from MIB-II family */
    RIP_PKT *  pRip       /* received RIP message */
)
{
    if (pRip->rip_vers == 1)
    {
        /*
         * The RFC specification recommends, but does not require, rejecting
         * @ version 1 packets when authentication is enabled.
         */
        return (ERROR);
    }
    /*
     * @ The authentication type field in the RIP message corresponds to
     * @ the first two bytes of the sa_data field overlaid on that
     * @ message by the sockaddr structure contained within the RIP_PKT
     * @ structure (see rip/ripLib.h).
     */
    if ( (pRip->rip_nets[0].rip_dst.sa_data[0] != 0) ||
         (pRip->rip_nets[0].rip_dst.sa_data[1] !=
          M2_rip2IfConfAuthType_simplePassword) )
    {
        /* Unrecognized authentication type. */
        return (ERROR);
    }
    /*
     * Discard version 2 packets requesting simple password authentication
     * @ which did not match the MIB variable.
     */
    return (ERROR);
}
```

A comparison against a different key could be performed as follows:

```
bzero ( (char *)&key, AUTHKEYLEN);      /* AUTHKEYLEN from rip/m2RipLib.h */
/*
 * @ The start of the authorization key corresponds to the third byte
 * @ of the sa_data field in the sockaddr structure overlaid on the
 * @ body of the RIP message by the RIP_PKT structure. It continues
 * @ for the final 14 bytes of that structure and the first two bytes
 * @ of the following rip_metric field.
 */
bcopy ( (char *) (pRip->rip_nets[0].rip_dst.sa_data + 2),
        (char *)&key, AUTHKEYLEN);
if (bcmp ( (char *) key, privateKey, AUTHKEYLEN) != 0)
```

```

    {
    /* Key does not match: reject message. */
    return (ERROR);
    }
return (OK);

```

The `ripAuthHookDelete()` routine will remove the installed function. If authentication is still enabled for the interface, all incoming messages that do not use simple password authentication will be rejected until a routine is provided.

**RETURNS** OK, if hook added; or **ERROR** otherwise.

**ERRNO** S\_m2Lib\_INVALID\_PARAMETER  
S\_m2Lib\_ENTRY\_NOT\_FOUND

**SEE ALSO** ripLib

---

## ripAuthHookDelete()

**NAME** ripAuthHookDelete() – remove an authentication hook from a RIP interface

**SYNOPSIS**

```

STATUS ripAuthHookDelete
(
    char* pIpAddr          /* IP address in dotted decimal notation */
)

```

**DESCRIPTION** This routine removes an assigned authentication hook from a registered interface indicated by *pIpAddr*. (Interfaces created or changed after a RIP session has started may be installed/updated with the `ripIfSearch()` and `ripIfReset()` routines). If authentication is still enabled for the interface, RIP-2 messages using simple password authentication will be accepted if they match the key in the MIB variable, but all other incoming messages will be rejected until a routine is provided.

**RETURNS** OK; or **ERROR**, if the interface could not be found.

**ERRNO** S\_m2Lib\_INVALID\_PARAMETER  
S\_m2Lib\_ENTRY\_NOT\_FOUND

**SEE ALSO** ripLib

## ripAuthKeyAdd()

**NAME** ripAuthKeyAdd() – add a new RIP authentication key

**SYNOPSIS**

```
STATUS ripAuthKeyAdd
(
    char * pInterfaceName,    /* interface to add a key */
    UINT16 keyId,            /* the keyId for this new key */
    char * pKey,              /* the secret key */
    UINT  keyLen,            /* length of the secret key */
    UINT  authProto,         /* auth protocol to use (1 = MD5) */
    ULONG timeValid          /* number of seconds until key expires */
)
```

**DESCRIPTION** This routine is used to add a new RIP authentication key for a specific interface.

**RETURNS** ERROR, if the interface does not exist, or the *keyId* already exists, or if the protocol is not supported; OK, if key was entered.

**SEE ALSO** ripLib

---

## ripAuthKeyDelete()

**NAME** ripAuthKeyDelete() – delete an existing RIP authentication key

**SYNOPSIS**

```
STATUS ripAuthKeyDelete
(
    char * pInterfaceName,    /* interface to delete a key from */
    UINT16 keyId              /* the keyId of the key to delete */
)
```

**DESCRIPTION** This routine is used to delete a RIP authentication key for a specific interface.

**RETURNS** ERROR, if the interface does not exist, or the *keyId* does not exist; OK, if key was deleted.

**SEE ALSO** ripLib

---

## ripAuthKeyFind()

**NAME** `ripAuthKeyFind()` – find a RIP authentication key

**SYNOPSIS**

```
STATUS ripAuthKeyFind
(
    struct interface * ifp, /* interface to search for key */
    UINT16          keyId, /* the keyId of the key to search for */
    RIP_AUTH_KEY * * pKey  /* storage for the key data */
)
```

**DESCRIPTION** This routine is used to find a RIP authentication key based on a specified interface and *keyId*. When a key is found, a pointer to the `RIP_AUTH_KEY` struct for the key is stored in *pKey*.

**RETURNS** `ERROR`, if the key is not found; `OK` if the key was found.

**SEE ALSO** `ripLib`

---

## ripAuthKeyFindFirst()

**NAME** `ripAuthKeyFindFirst()` – find a RIP authentication key

**SYNOPSIS**

```
STATUS ripAuthKeyFindFirst
(
    struct interface * ifp, /* interface to search for key */
    RIP_AUTH_KEY * * pKey  /* storage for the key data */
)
```

**DESCRIPTION** This routine is used to find a RIP authentication key based on a specified interface. Because a *keyId* is not specified, this routine returns the first non-expired key found for the interface. When a key is found, a pointer to the `RIP_AUTH_KEY` structure for the key is returned in *pKey*.

**RETURNS** `ERROR`, if a key is not found; `OK`, if a key was found.

**SEE ALSO** `ripLib`

---

## ripAuthKeyInMD5()

**NAME** ripAuthKeyInMD5() – authenticate an incoming RIP-2 message using MD5

**SYNOPSIS**

```
STATUS ripAuthKeyInMD5
(
    struct interface * ifp, /* interface message received on */
    RIP_PKT *        pRip, /* received RIP message */
    UINT             size /* length of the RIP message */
)
```

**DESCRIPTION** This routine is used to authenticate an incoming RIP-2 message using the MD5 digest protocol. This authentication scheme is described in RFC 2082.

**RETURNS** ERROR, if could not authenticate; OK, if authenticated.

**SEE ALSO** ripLib

---

## ripAuthKeyOut1MD5()

**NAME** ripAuthKeyOut1MD5() – start MD5 authentication of an outgoing RIP-2 message

**SYNOPSIS**

```
STATUS ripAuthKeyOut1MD5
(
    struct interface * pIfp, /* interface message being sent on */
    struct netinfo *  pNetinfo, /* pointer to next RIP entry to fill in */
    RIP2_AUTH_PKT_HDR * * ppAuthHdr, /* stores the authentication header */
    RIP_AUTH_KEY * * ppAuthKey /* stores the authentication key to use */
)
```

**DESCRIPTION** This routine is used to start the authentication of an outgoing RIP-2 message by adding the authentication header used for MD5 authentication. This authentication scheme is described in RFC 2082. This function returns a pointer the authentication header and a pointer to the looked up authentication key.

**RETURNS** ERROR, if a key could not be found; OK, if the header was added.

**SEE ALSO** ripLib

---

## ripAuthKeyOut2MD5( )

**NAME** ripAuthKeyOut2MD5( ) – authenticate an outgoing RIP-2 message using MD5

**SYNOPSIS**

```
void ripAuthKeyOut2MD5
(
    RIP_PKT *      pRip,      /* RIP message to authenticate */
    UINT *        pSize,     /* length of the RIP message */
    struct netinfo * pNetinfo, /* pointer to next RIP entry to fill in */
    RIP2_AUTH_PKT_HDR * pAuthHdr, /* pointer to auth header in the message */
    RIP_AUTH_KEY *  pAuthKey /* the auth key data to use */
)
```

**DESCRIPTION** This routine is used to authenticate an outgoing RIP-2 message using the MD5 digest protocol. This authentication scheme is described in RFC 2082. This function modifies the size given in *pSize* to account for the extra **auth** trailer data. The **auth** trailer is appended to the given *RIP\_PKT* and the authentication digest is filled in.

**RETURNS** N/A

**SEE ALSO** ripLib

---

## ripAuthKeyShow( )

**NAME** ripAuthKeyShow( ) – show current authentication configuration

**SYNOPSIS**

```
void ripAuthKeyShow
(
    UINT showKey /* if non-zero then key values are shown */
)
```

**DESCRIPTION** This routines shows the current configuration of the authentication keys for each interface.

**RETURNS** N/A

**SEE ALSO** ripLib

## ripDebugLevelSet( )

**NAME** ripDebugLevelSet( ) – specify amount of debugging output

**SYNOPSIS**

```
void ripDebugLevelSet
(
    int level          /* verbosity level (0 - 3) */
)
```

**DESCRIPTION** This routine determines the amount of debugging information sent to standard output during the RIP session. Higher values of the *level* parameter result in increasingly verbose output. A *level* of zero restores the default behavior by disabling all debugging output.

**RETURNS** N/A

**ERRNO** N/A

**SEE ALSO** ripLib

---

## ripFilterDisable( )

**NAME** ripFilterDisable( ) – prevent strict border gateway filtering

**SYNOPSIS**

```
void ripFilterDisable (void)
```

**DESCRIPTION** This routine configures an active RIP session to ignore the restrictions necessary for RIP-1 and RIP-2 routers to operate correctly in the same network. All border gateway filtering is ignored and all routes to subnets, supernets, and specific hosts will be sent over any available interface. This operation is only correct if no RIP-1 routers are present anywhere on the network. Results are unpredictable if that condition is not met, but high rates of packet loss and widespread routing failures are likely.

The border gateway filtering rules are in force by default.

**RETURNS** N/A

**ERRNO** N/A

**SEE ALSO** ripLib

---

## ripFilterEnable()

<b>NAME</b>	<code>ripFilterEnable()</code> – activate strict border gateway filtering
<b>SYNOPSIS</b>	<code>void ripFilterEnable (void)</code>
<b>DESCRIPTION</b>	This routine configures an active RIP session to enforce the restrictions necessary for RIP-1 and RIP-2 routers to operate correctly in the same network as described in section 3.2 of RFC 1058 and section 3.3 of RFC 1723. When enabled, routes to portions of a logical network (including host routes) are limited to routers within that network. Updates sent outside that network include only a single entry representing the entire network. That entry subsumes all subnets and host-specific routes. If supernets are used, the entry advertises the largest class-based portion of the supernet reachable through the connected interface.
<b>RETURNS</b>	N/A
<b>ERRNO</b>	N/A
<b>SEE ALSO</b>	<code>ripLib</code>

---

## ripIfExcludeListAdd()

<b>NAME</b>	<code>ripIfExcludeListAdd()</code> – add an interface to the RIP exclusion list
<b>SYNOPSIS</b>	<pre>STATUS ripIfExcludeListAdd (     char * pIfName          /* name of interface to be excluded */ )</pre>
<b>DESCRIPTION</b>	This function adds the interface specified by <i>ifName</i> to a list of interfaces on which RIP will not be started. This can be used to prevent RIP from starting on an interface.
<b>RETURNS</b>	OK if the interface was successfully added to the list; <b>ERROR</b> otherwise.
	<b>NOTE:</b> This command must be issued prior to the interface being added to the system, as RIP starts on an interface, unless it has been excluded, as soon as an interface comes up. If RIP was already running on the interface which is now desired to be excluded from RIP, the <code>ripIfReset()</code> command should be used after the <code>ripIfExcludeListAdd()</code> command.

---

SEE ALSO [ripLib](#)

---

## ripIfExcludeListDelete()

**NAME** `ripIfExcludeListDelete()` – delete an interface from RIP exclusion list

**SYNOPSIS**

```
STATUS ripIfExcludeListDelete  
(  
    char * pIfName          /* name of un-excluded interface */  
)
```

**DESCRIPTION** This function deletes the interface specified by *ifName* from the list of interfaces on which RIP will not be started. That is, RIP will start on the interface when it is added or comes up.

**RETURNS** OK if the interface was successfully removed from the list;  
ERROR otherwise.

---

**NOTE:** RIP will not automatically start on the interface. The `ripIfSearch()` call will need to be made after this call to cause RIP to start on this interface.

---

SEE ALSO [ripLib](#)

---

## ripIfExcludeListShow()

**NAME** `ripIfExcludeListShow()` – show the RIP interface exclusion list

**SYNOPSIS**

```
void ripIfExcludeListShow (void)
```

**DESCRIPTION** This function prints out the interfaces on which RIP will not be started.

**RETURNS** Nothing

SEE ALSO [ripLib](#)

---

## ripIfReset()

<b>NAME</b>	<b>ripIfReset()</b> – alter the RIP configuration after an interface changes
<b>SYNOPSIS</b>	<pre><b>STATUS</b> ripIfReset (     <b>char</b> * pIfName          /* name of changed interface */ )</pre>
<b>DESCRIPTION</b>	This routine updates the interface list and routing tables to reflect address and/or netmask changes for the device indicated by <i>pIfName</i> . To accommodate possible changes in the network number, all routes using the named interface are removed from the routing tables, but will be added in the next route update if appropriate. None of the removed routes are poisoned, so it may take some time for the routing tables of all the RIP participants to stabilize if the network number has changed. This routine replaces the existing interface structure with a new one. Thus, any interface specific MIB2 changes that were made to the interface being reset will be lost
<b>RETURNS</b>	OK, or <b>ERROR</b> if named interface not found or not added to list.
<b>ERRNO</b>	N/A
<b>SEE ALSO</b>	<b>ripLib</b>

---

## ripIfSearch()

<b>NAME</b>	<b>ripIfSearch()</b> – add new interfaces to the internal list
<b>SYNOPSIS</b>	<pre><b>void</b> ripIfSearch (void)</pre>
<b>DESCRIPTION</b>	By default, a RIP session will not recognize any interfaces initialized after it has started. This routine schedules a search for additional interfaces that will occur during the next update of the internal routing table. Once completed, the session will accept and send RIP messages over the new interfaces.
<b>RETURNS</b>	N/A
<b>ERRNO</b>	N/A
<b>SEE ALSO</b>	<b>ripLib</b>

## ripIfShow()

<b>NAME</b>	<b>ripIfShow()</b> – display the internal interface table maintained by RIP
<b>SYNOPSIS</b>	<code>void ripIfShow (void)</code>
<b>DESCRIPTION</b>	This routine prints every entry in the local RIP interface table. The interface name, interface index, the UP/DOWN status and the interface address and netmask are displayed.
<b>RETURNS</b>	N/A
<b>ERRNO</b>	N/A
<b>SEE ALSO</b>	<b>ripLib</b>

---

## ripLeakHookAdd()

<b>NAME</b>	<b>ripLeakHookAdd()</b> – add a hook to bypass the RIP and kernel routing tables
<b>SYNOPSIS</b>	<pre>STATUS ripLeakHookAdd (     char * pIpAddr,          /* IP address in dotted decimal notation */     FUNCPTR pLeakHook       /* function pointer to hook */ )</pre>
<b>DESCRIPTION</b>	<p>This routine installs a hook routine to support alternative routing protocols for the registered interface given by <i>pIpAddr</i>. (Interfaces created or changed after a RIP session has started may be installed/updated with the <b>ripIfSearch()</b> and <b>ripIfReset()</b> routines).</p> <p>The hook uses the following interface:</p> <pre>STATUS ripLeakHookRtn (long dest, long gateway, long netmask)</pre> <p>The RIP session will not add the given route to any tables if the hook routine returns <b>OK</b>, but will create a route entry otherwise.</p> <p>The <b>ripLeakHookDelete()</b> will allow the RIP session to add new routes unconditionally.</p>
<b>RETURNS</b>	<b>OK</b> ; or <b>ERROR</b> , if the interface could not be found.

**ERRNO** S\_m2Lib\_INVALID\_PARAMETER  
S\_m2Lib\_ENTRY\_NOT\_FOUND

**SEE ALSO** ripLib

---

## ripLeakHookDelete()

**NAME** ripLeakHookDelete() – remove a table bypass hook from a RIP interface

**SYNOPSIS**

```
STATUS ripLeakHookDelete
(
    char* pIpAddr          /* IP address in dotted decimal notation */
)
```

**DESCRIPTION** This routine removes the assigned bypass hook from a registered interface indicated by *pIpAddr*. (Interfaces created or changed after a RIP session has started may be installed/updated with the **ripIfSearch()** and **ripIfReset()** routines). The RIP session will return to the default behavior and add entries to the internal RIP table and kernel routing table unconditionally.

**RETURNS** OK; or ERROR, if the interface could not be found.

**ERRNO** S\_m2Lib\_INVALID\_PARAMETER  
S\_m2Lib\_ENTRY\_NOT\_FOUND

**SEE ALSO** ripLib

---

## ripLibInit()

**NAME** ripLibInit() – initialize the RIP routing library

**SYNOPSIS**

```
STATUS ripLibInit
(
    BOOL supplier,          /* operate in silent mode? */
    BOOL gateway,          /* act as gateway to the Internet? */
    BOOL multicast,        /* use multicast or broadcast addresses? */
    int version,           /* 1 or 2: selects format of outgoing messages */
    int timerRate,         /* update frequency for internal routing table */
)
```

**ripLibInit()**

```

    int  supplyInterval, /* update frequency for neighboring routers */
    int  expire,         /* maximum interval for renewing learned routes */
    int  garbage,       /* elapsed time before deleting stale route */
    int  authType       /* default authentication type to use */
)

```

**DESCRIPTION**

This routine creates and initializes the global data structures used by the RIP routing library and starts a RIP session to maintain routing tables for a host. You must call **ripLibInit()** before you can use any other **ripLib** routines. A VxWorks image automatically invokes **ripLibInit()** if **INCLUDE\_RIP** was defined when the image was built.

The resulting RIP session will monitor all network interfaces that are currently available for messages from other RIP routers. If the *supplier* parameter is true, it will also respond to specific requests from other routers and transmit route updates over every known interface at the interval specified by *supplyInterval*.

Specifying a *gateway* setting of true establishes this router as a gateway to the wider Internet, capable of routing packets anywhere within the local networks. The final *multicast* flag indicates whether the RIP messages are sent to the pre-defined multicast address of 224.0.0.9 (which requires a *version* setting of 2) or to the broadcast address of the interfaces.

The *version* parameter determines the format used for outgoing RIP messages, and also sets the initial settings of the MIB-II compatibility switches in combination with the *multicast* flag. A *version* of 1 will restrict all incoming traffic to that older message type. A *version* of 2 will set the receive switch to accept either type unless *multicast* is true, which limits reception to version 2 messages only. SNMP agents may alter those settings on a per-interface basis once startup is complete.

The remaining parameters set various system timers used to maintain the routing table. All of the values are expressed in seconds, and must be greater than or equal to 1. The *timerRate* determines how often the routing table is examined for changes and expired routes. The *supplyInterval* must be an exact multiple of that value. The *expire* parameter specifies the maximum time between updates before a route is invalidated and removed from the kernel table. Expired routes are then deleted from the internal RIP routing table if no update has been received within the time set by the *garbage* parameter.

The following configuration parameters determine the initial values for all these settings. The default timer values match the settings indicated in the RFC specification.

Parameter Name	Default Value	Configuration Parameter
<i>supplier</i>	0 (FALSE)	RIP_SUPPLIER
<i>gateway</i>	0 (FALSE)	RIP_GATEWAY
<i>multicast</i>	0 (FALSE)	RIP_MULTICAST
<i>version</i>	1	RIP_VERSION
<i>timerRate</i>	1	RIP_TIMER_RATE

Parameter Name	Default Value	Configuration Parameter
<i>supplyInterval</i>	30	RIP_SUPPLY_INTERVAL
<i>expire</i>	180	RIP_EXPIRE_TIME
<i>garbage</i>	300	RIP_GARBAGE_TIME
<i>authType</i>	1	RIP_AUTH_TYPE

**RETURNS** OK; or **ERROR**, if configuration fails.

**SEE ALSO** **ripLib**

---

## ripRouteHookAdd()

**NAME** **ripRouteHookAdd()** – add a hook to install static and non-RIP routes into RIP

**SYNOPSIS**

```
STATUS ripRouteHookAdd
(
    FUNCPTR pRouteHook    /* function pointer to hook */
)
```

**DESCRIPTION** This routine installs a hook routine that you can use to give RIP the ability to respond to route-add events generated by non-RIP agents. By design, RIP is not interested in the routes generated by non-RIP agents. If you do not install a route hook function, RIP continues this default behavior. However, if you want RIP to add these non-RIP routes to its internal routing database and even propagate routes added by other agents, you must use **ripRouteHookAdd()** to register a function of the form:

```
STATUS YourRipRouteHookRtn
(
    struct ROUTE_INFO * pRouteInfo,
    int protoId,
    BOOL primaryRoute,
    int flags
)
```

RIP invokes this function in response to the following events:

1. A non-RIP non-system route was added to the routing table.
2. A route change message arrived.
3. An ICMP redirect message arrived.

The returned function value of the route hook routine tells rip how to respond to the event. In the first case, the returned function value tells RIP whether to add or ignore the

new route. In the second case, the returned function tells RIP whether to delete the specified route or change its metric. In the third case, the event is of no direct importance for RIP, so RIP ignores the returned value of the route hook function.

*pRouteInfo*

This parameter passes in a pointer to a route information structure that stores the routing message. You should not access the contents of this structure directly. Instead, use **ripAddrXtract()** to extract the following information:

- destination address
- netmask
- gateway address
- old gateway address (if available)

*protoId*

This parameter passes in the ID of the protocol that generated the event. Valid protocol IDs are defined in **m2Lib.h** as follows:

- M2\_ipRouteProto\_other** (static routes)
- M2\_ipRouteProto\_local**
- M2\_ipRouteProto\_netmgmt**
- M2\_ipRouteProto\_icmp**
- M2\_ipRouteProto\_egp**
- M2\_ipRouteProto\_ggp**
- M2\_ipRouteProto\_hello**
- M2\_ipRouteProto\_rip**
- M2\_ipRouteProto\_is\_is**
- M2\_ipRouteProto\_es\_is**
- M2\_ipRouteProto\_ciscoIgrp**
- M2\_ipRouteProto\_bbnSpfIgp**
- M2\_ipRouteProto\_ospf**
- M2\_ipRouteProto\_bgp**

*primaryRoute*

This parameter passes in a boolean value that indicates whether the route is a primary route. **TRUE** indicates a primary route. **FALSE** indicates a duplicate route.

*flags*

This parameter passes in a value that indicates which event occurred:

**0** (zero)

This indicates a route added to the routing table by a non-RIP agent.

**RIP\_ROUTE\_CHANGE\_REC'D**

This indicates a route change message.

**RIP\_REDIRECT\_REC'D**

This indicates an ICMP redirect message.

#### A New Non-RIP Non-System Route was Added to the Routing Table

In response to this event, RIP needs to be told whether to ignore or add the route. RIP does this on the basis of the returned function value of the route hook routine. In the case of route-add event, RIP interprets the returned function value of the route hook routine as the metric for the route.

If the metric is `HOPCNT_INFINITY`, RIP ignores the route. If the metric is greater than zero but less than `HOPCNT_INFINITY`, RIP considers the route for inclusion. If the route is new to RIP, RIP adds the new route to its internal database, and then propagates the route in its subsequent update messages. If RIP already stores a route for that destination, RIP compares the metric of the new route and the stored route. If the new route has a better (lower) metric, RIP adds the new route. Otherwise, RIP ignores the new route.

When generating its returned function value, your route hook routine can use the creator of the event (*protoID*) as a factor in the decision on whether to include the route. For example, if you wanted the route hook to tell RIP to ignore all non-RIP routes except static routes, your route hook would return `HOPCNT_INFINITY` if the *protoID* were anything other than `M2_ipRouteProto_other`. Thus, your route hook routine is a vehicle through which you can implement a policy for including non-RIP routes in the RIP internal route data base.

When designing your policy, you should keep in mind how RIP prioritizes these non-RIP routes and when it deletes these non-RIP routes. For example, non-RIP routes never time out. They remain in the RIP database until one of the following events occurs:

1. An agent deletes the route from the system routing table.
2. An agent deletes the interface through which the route passes.
3. A route change message for the route arrives.

Also, these non-RIP routes take precedence over RIP routes to the same destination. RIP ignores routes learned from RIP peers if a route to the same destination was recommended by the hook routine. This non-RIP route takes precedence over the RIP route without regard of the route metric. However, if the route hook routine adds multiple same-destination routes, the route with the lowest metric takes precedence. If the route hook route approves multiple same-metric same-destination routes, the most recently added route is installed.

#### A Route Change Notification Arrived

In response to this event, RIP needs to be told whether to delete the route or change its metric. If the hook returns a value greater than or equal to `HOPCNT_INFINITY`, RIP deletes the route from its internal routing data base. If the hook routine returns a valid metric (a value greater than zero but less than `HOPCNT_INFINITY`), RIP reassigns the routes metric to equal the returned value of the route hook routine. If the returned value of the route hook route is invalid (less than zero) RIP ignores the event. RIP also ignores the event if the route specified in *pRouteInfo* is not one stored in its internal data base.

#### An ICMP Redirect Message Arrived

In response to this event, RIP never needs to make any changes to its internal routing

database. Thus, RIP ignores the returned function value of the route hook routine called in response to an ICMP redirect message. However, if the event is of interest to your particular environment, and it makes sense to catch the event in the context of the RIP task, you can use the route hook routine to do so.

Within your route hook routine, you can recognize an ICMP event by checking whether the flags parameter value sets the **RIP\_REDIRECT\_REC'D** bit. The *primaryRoute* parameter passes in a boolean value that indicates whether the route is primary route. If the *primaryRoute* passes in **FALSE**, the route hook routine need will most likely need to do nothing more. If this parameter passes in **TRUE**, take whatever action (if any) that you know to be appropriate to your particular environment.

**RETURNS** OK; or **ERROR**, if RIP is not initialized.

**SEE ALSO** ripLib

---

## ripRouteHookDelete()

**NAME** ripRouteHookDelete() – remove the route hook

**SYNOPSIS** **STATUS** ripRouteHookDelete (void)

**DESCRIPTION** This routine removes the route hook installed earlier by the **ripRouteHookAdd()** routine. This will cause RIP to ignore any routes added to the system Routing database.

**RETURNS** OK; or **ERROR**, if RIP is not initialized.

**SEE ALSO** ripLib

---

## ripRouteShow()

**NAME** ripRouteShow() – display the internal routing table maintained by RIP

**SYNOPSIS** void ripRouteShow (void)

**DESCRIPTION** This routine prints every entry in the local RIP routing table. The flags displayed below the destination, gateway, and netmask addresses indicate the current route status. Entries with the **RTS\_INTERFACE** flag indicate locally generated routes to directly connected

networks. If `RTS_SUBNET` is set for an entry, it is subject to border gateway filtering (if enabled). When `RTS_INTERNAL` is also present, the corresponding entry is an “artificial” route created to supply distant networks with legitimate destinations if border filtering excludes the actual entry. Those entries are not copied to the kernel routing table. The `RTS_CHANGED` flag marks entries added or modified in the last timer interval that will be included in a triggered update. The `RTS_OTHER` flag is set for routes learnt from other sources. The `RTS_PRIMARY` flag (set only if the `RTS_OTHER` flag is also set) indicates that the route is a primary route, visible to the IP forwarding process. The `DOWN` flag indicates that the interface through which the gateway is reachable is down.

SEE ALSO `ripLib`

---

## ripSendHookAdd()

NAME `ripSendHookAdd()` – add an update filter to a RIP interface

SYNOPSIS

```
STATUS ripSendHookAdd
(
    char* pIpAddr,           /* IP address in dotted decimal notation */
    BOOL (* ripSendHook) (struct rt_entry* pRt)
                          /* Routine to use. */
)
```

DESCRIPTION This routine installs a hook routine to screen individual route entries for inclusion in a periodic update. The routine is installed for the registered interface given by *pIpAddr*. (Interfaces created or changed after a RIP session has started may be installed/updated with the `ripIfSearch()` and `ripIfReset()` routines).

The hook uses the following prototype:

```
BOOL ripSendHookRtn (struct rt_entry* pRt);
```

If the hook returns `FALSE`, the route is not included in the update. Otherwise, it is included if it meets the other restrictions, such as simple split horizon and border gateway filtering. The `ripSendHookDelete()` routine removes this additional filter from the output processing.

RETURNS `OK`; or `ERROR`, if the interface could not be found.

ERRNO `S_m2Lib_INVALID_PARAMETER`  
`S_m2Lib_ENTRY_NOT_FOUND`

SEE ALSO `ripLib`

## ripSendHookDelete()

<b>NAME</b>	<b>ripSendHookDelete()</b> – remove an update filter from a RIP interface
<b>SYNOPSIS</b>	<pre><b>STATUS</b> ripSendHookDelete (     <b>char*</b> pIpAddr           /* IP address in dotted decimal notation */ )</pre>
<b>DESCRIPTION</b>	This routine removes the hook routine that allowed additional screening of route entries in periodic updates from the registered interface indicated by <i>pIpAddr</i> . (Interfaces created or changed after a RIP session has started may be installed/updated with the <b>ripIfSearch()</b> and <b>ripIfReset()</b> routines). The RIP session will return to the default behavior and include any entries that meet the other restrictions (such as simple split horizon).
<b>RETURNS</b>	OK; or <b>ERROR</b> , if the interface could not be found.
<b>ERRNO</b>	<b>S_m2Lib_INVALID_PARAMETER</b> <b>S_m2Lib_ENTRY_NOT_FOUND</b>
<b>SEE ALSO</b>	<b>ripLib</b>

---

## ripShutdown()

<b>NAME</b>	<b>ripShutdown()</b> – terminate all RIP processing
<b>SYNOPSIS</b>	<pre><b>STATUS</b> ripShutdown (void)</pre>
<b>DESCRIPTION</b>	This routine “poisons” all routes in the current table by transmitting updates with an infinite metric for each entry over all available interfaces. It then halts all RIP processing and removes the associated tasks and data structures. When completed successfully, the RIP services are unavailable until restarted with the <b>ripLibInit()</b> routine.
<b>RETURNS</b>	OK if shutdown completed, or <b>ERROR</b> otherwise.
<b>ERRNO</b>	N/A
<b>SEE ALSO</b>	<b>ripLib</b>

---

## rlogin()

<b>NAME</b>	<b>rlogin()</b> – log in to a remote host
<b>SYNOPSIS</b>	<pre>STATUS rlogin (     char * host          /* name of host to connect to */ )</pre>
<b>DESCRIPTION</b>	<p>This routine allows users to log in to a remote host. It may be called from the VxWorks shell as follows:</p> <pre>-&gt; rlogin "remoteSystem"</pre> <p>where <i>remoteSystem</i> is either a host name, which has been previously added to the remote host table by a call to <b>hostAdd()</b>, or an Internet address in dot notation (e.g., "90.0.0.2"). The remote system will be logged into with the current user name as set by a call to <b>iam()</b>. The user disconnects from the remote system by typing:</p> <pre>~.</pre> <p>as the only characters on the line, or by simply logging out from the remote system using <b>logout()</b>.</p>
<b>RETURNS</b>	OK, or <b>ERROR</b> if the host is unknown, no privileged ports are available, the routine is unable to connect to the host, or the child process cannot be spawned.
<b>SEE ALSO</b>	<b>rlogLib</b> , <b>iam()</b> , <b>logout()</b>

---

## rlogind()

<b>NAME</b>	<b>rlogind()</b> – the VxWorks remote login daemon
<b>SYNOPSIS</b>	<pre>void rlogind (void)</pre>
<b>DESCRIPTION</b>	<p>This routine provides a facility for remote users to log in to VxWorks over the network. If <b>INCLUDE_RLOGIN</b> is defined, <b>rlogind()</b> is spawned by <b>rlogInit()</b> at boot time.</p> <p>Remote login requests will cause <b>stdin</b>, <b>stdout</b>, and <b>stderr</b> to be directed away from the console. When the remote user disconnects, <b>stdin</b>, <b>stdout</b>, and <b>stderr</b> are restored, and the shell is restarted. The <b>rlogind()</b> routine uses the remote user verification protocol</p>

specified by the UNIX remote shell daemon documentation, but ignores all the information except the user name, which is used to set the VxWorks remote identity (see the manual entry for **iam()**).

The remote login daemon requires the existence of a pseudo-terminal device, which is created by **rlogInit()** before **rlogind()** is spawned. The **rlogind()** routine creates two child processes, **tRlogInTask** and **tRlogOutTask**, whenever a remote user is logged in. These processes exit when the remote connection is terminated.

#### **VXWORKS AE PROTECTION DOMAINS**

Under VxWorks AE, you can call this function from within the kernel protection domain only. In addition, all arguments to this function can reference only that data which is valid in the kernel protection domain. This restriction does not apply under non-AE versions of VxWorks.

**RETURNS** N/A

**SEE ALSO** **rlogLib**, **rlogInit()**, **iam()**

---

## **rlogInit()**

**NAME** **rlogInit()** – initialize the remote login facility

**SYNOPSIS** **STATUS rlogInit (void)**

**DESCRIPTION** This routine initializes the remote login facility. It creates a pty (pseudo tty) device and spawns **rlogind()**. If **INCLUDE\_RLOGIN** is included, **rlogInit()** is called automatically at boot time.

#### **VXWORKS AE PROTECTION DOMAINS**

Under VxWorks AE, you can call this function from within the kernel protection domain only. This restriction does not apply under non-AE versions of VxWorks.

**RETURNS** OK or ERROR.

**SEE ALSO** **rlogLib**, **ptyDrv**

---

## rm()

<b>NAME</b>	<code>rm()</code> – remove a file
<b>SYNOPSIS</b>	<pre>STATUS rm (     const char * fileName    /* name of file to remove */ )</pre>
<b>DESCRIPTION</b>	This command is provided for UNIX similarity. It simply calls <code>remove()</code> .
<b>RETURNS</b>	OK, or <code>ERROR</code> if the file cannot be removed.
<b>SEE ALSO</b>	<code>usrFsLib</code> , <code>remove()</code> , <i>VxWorks Programmer's Guide: Target Shell</i>

---

## rmdir()

<b>NAME</b>	<code>rmdir()</code> – remove a directory
<b>SYNOPSIS</b>	<pre>STATUS rmdir (     const char * dirName    /* name of directory to remove */ )</pre>
<b>DESCRIPTION</b>	<p>This command removes an existing directory from a hierarchical file system. The <i>dirName</i> string specifies the name of the directory to be removed, and may be either a full or relative pathname.</p> <p>This call is supported by the VxWorks NFS and dosFs file systems.</p>
<b>RETURNS</b>	OK, or <code>ERROR</code> if the directory cannot be removed.
<b>SEE ALSO</b>	<code>usrFsLib</code> , <code>mkdir()</code> , <i>VxWorks Programmer's Guide: Target Shell</i>

## rngBufGet()

**NAME** `rngBufGet()` – get characters from a ring buffer

**SYNOPSIS**

```
int rngBufGet
(
    RING_ID rngId,          /* ring buffer to get data from */
    char *  buffer,        /* pointer to buffer to receive data */
    int     maxbytes       /* maximum number of bytes to get */
)
```

**DESCRIPTION** This routine copies bytes from the ring buffer *rngId* into *buffer*. It copies as many bytes as are available in the ring, up to *maxbytes*. The bytes copied will be removed from the ring.

**RETURNS** The number of bytes actually received from the ring buffer; it may be zero if the ring buffer is empty at the time of the call.

**SEE ALSO** `rngLib`

---

## rngBufPut()

**NAME** `rngBufPut()` – put bytes into a ring buffer

**SYNOPSIS**

```
int rngBufPut
(
    RING_ID rngId,          /* ring buffer to put data into */
    char *  buffer,        /* buffer to get data from */
    int     nbytes         /* number of bytes to try to put */
)
```

**DESCRIPTION** This routine puts bytes from *buffer* into ring buffer *ringId*. The specified number of bytes will be put into the ring, up to the number of bytes available in the ring.

**RETURNS** The number of bytes actually put into the ring buffer; it may be less than number requested, even zero, if there is insufficient room in the ring buffer at the time of the call.

**SEE ALSO** `rngLib`

---

## rngCreate()

<b>NAME</b>	<code>rngCreate()</code> – create an empty ring buffer
<b>SYNOPSIS</b>	<pre>RING_ID rngCreate (     int nbytes           /* number of bytes in ring buffer */ )</pre>
<b>DESCRIPTION</b>	This routine creates a ring buffer of size <i>nbytes</i> , and initializes it. Memory for the buffer is allocated from the system memory partition.
<b>RETURNS</b>	The ID of the ring buffer, or NULL if memory cannot be allocated.
<b>SEE ALSO</b>	<code>rngLib</code>

---

## rngDelete()

<b>NAME</b>	<code>rngDelete()</code> – delete a ring buffer
<b>SYNOPSIS</b>	<pre>void rngDelete (     RING_ID ringId      /* ring buffer to delete */ )</pre>
<b>DESCRIPTION</b>	This routine deletes a specified ring buffer. Any data currently in the buffer will be lost.
<b>RETURNS</b>	N/A
<b>SEE ALSO</b>	<code>rngLib</code>

## rngFlush()

<b>NAME</b>	<b>rngFlush()</b> – make a ring buffer empty
<b>SYNOPSIS</b>	<pre>void rngFlush (     RING_ID ringId          /* ring buffer to initialize */ )</pre>
<b>DESCRIPTION</b>	This routine initializes a specified ring buffer to be empty. Any data currently in the buffer will be lost.
<b>RETURNS</b>	N/A
<b>SEE ALSO</b>	<b>rngLib</b>

---

## rngFreeBytes()

<b>NAME</b>	<b>rngFreeBytes()</b> – determine the number of free bytes in a ring buffer
<b>SYNOPSIS</b>	<pre>int rngFreeBytes (     RING_ID ringId          /* ring buffer to examine */ )</pre>
<b>DESCRIPTION</b>	This routine determines the number of bytes currently unused in a specified ring buffer.
<b>RETURNS</b>	The number of unused bytes in the ring buffer.
<b>SEE ALSO</b>	<b>rngLib</b>

---

## rngIsEmpty()

**NAME** `rngIsEmpty()` – test if a ring buffer is empty

**SYNOPSIS**

```
BOOL rngIsEmpty
(
    RING_ID ringId          /* ring buffer to test */
)
```

**DESCRIPTION** This routine determines if a specified ring buffer is empty.

**RETURNS** TRUE if empty, FALSE if not.

**SEE ALSO** `rngLib`

---

## rngIsFull()

**NAME** `rngIsFull()` – test if a ring buffer is full (no more room)

**SYNOPSIS**

```
BOOL rngIsFull
(
    RING_ID ringId          /* ring buffer to test */
)
```

**DESCRIPTION** This routine determines if a specified ring buffer is completely full.

**RETURNS** TRUE if full, FALSE if not.

**SEE ALSO** `rngLib`

## rngMoveAhead()

<b>NAME</b>	<b>rngMoveAhead()</b> – advance a ring pointer by <i>n</i> bytes
<b>SYNOPSIS</b>	<pre>void rngMoveAhead (     RING_ID ringId,      /* ring buffer to be advanced */     int      n           /* number of bytes ahead to move input pointer */ )</pre>
<b>DESCRIPTION</b>	This routine advances the ring buffer input pointer by <i>n</i> bytes. This makes <i>n</i> bytes available in the ring buffer, after having been written ahead in the ring buffer with <b>rngPutAhead()</b> .
<b>RETURNS</b>	N/A
<b>SEE ALSO</b>	<b>rngLib</b>

---

## rngNBytes()

<b>NAME</b>	<b>rngNBytes()</b> – determine the number of bytes in a ring buffer
<b>SYNOPSIS</b>	<pre>int rngNBytes (     RING_ID ringId      /* ring buffer to be enumerated */ )</pre>
<b>DESCRIPTION</b>	This routine determines the number of bytes currently in a specified ring buffer.
<b>RETURNS</b>	The number of bytes filled in the ring buffer.
<b>SEE ALSO</b>	<b>rngLib</b>

---

## rngPutAhead()

**NAME** `rngPutAhead()` – put a byte ahead in a ring buffer without moving ring pointers

**SYNOPSIS**

```
void rngPutAhead
(
    RING_ID ringId,          /* ring buffer to put byte in */
    char   byte,            /* byte to be put in ring */
    int    offset           /* offset beyond next input byte where to */
                           /* put byte */
)
```

**DESCRIPTION** This routine writes a byte into the ring, but does not move the ring buffer pointers. Thus the byte will not yet be available to `rngBufGet()` calls. The byte is written *offset* bytes ahead of the next input location in the ring. Thus, an offset of 0 puts the byte in the same position as `RNG_ELEM_PUT` would, except that the input pointer is not updated.

Bytes written ahead in the ring buffer with this routine can be made available all at once by subsequently moving the ring buffer pointers with the routine `rngMoveAhead()`.

Before calling `rngPutAhead()`, the caller must verify that at least *offset* + 1 bytes are available in the ring buffer.

**RETURNS** N/A

**SEE ALSO** `rngLib`

---

## romStart()

**NAME** `romStart()` – generic ROM initialization

**SYNOPSIS**

```
void romStart
(
    int startType           /* start type */
)
```

**DESCRIPTION** This is the first C code executed after reset.

This routine is called by the assembly start-up code in `romInit()`. It clears memory, copies ROM to RAM, and possibly invokes the uncompressor. It then jumps to the entry point of the uncompressed object code.

**RETURNS** N/A

**SEE ALSO** **bootInit**

---

## **round()**

**NAME** **round()** – round a number to the nearest integer

**SYNOPSIS**

```
double round
(
    double x /* value to round */
)
```

**DESCRIPTION** This routine rounds a double-precision value *x* to the nearest integral value.

**INCLUDE FILES** **math.h**

**RETURNS** The double-precision representation of *x* rounded to the nearest integral value.

**SEE ALSO** **mathALib**

---

## **roundf()**

**NAME** **roundf()** – round a number to the nearest integer

**SYNOPSIS**

```
float roundf
(
    float x /* argument */
)
```

**DESCRIPTION** This routine rounds a single-precision value *x* to the nearest integral value.

**INCLUDE FILES** **math.h**

**RETURNS** The single-precision representation of *x* rounded to the nearest integral value.

**SEE ALSO** **mathALib**

---

## routeAdd()

**NAME** routeAdd() – add a route

**SYNOPSIS**

```
STATUS routeAdd
(
    char * destination, /* inet addr or name of route destination */
    char * gateway      /* inet addr or name of gateway to destination */
)
```

**DESCRIPTION** This routine adds gateways to the network routing tables. It is called from a VxWorks machine that needs to establish a gateway to a destination network (or machine).

You can specify both *destination* and *gateway* in standard Internet address format (for example, 90.0.0.2), or you can specify them using their host names, as specified with **hostAdd()**.

This routine can be used to add multiple routes to the same destination differing by the gateway.

**EXAMPLE** Consider the following example:

```
-> routeAdd "90.0.0.0", "gate"
```

This call tells VxWorks that the machine with the host name “gate” is the gateway to network 90.0.0.0. The host “gate” must already have been created by **hostAdd()**.

Consider the following example:

```
-> routeAdd "90.0.0.0", "91.0.0.3"
```

This call tells VxWorks that the machine with the Internet address 91.0.0.3 is the gateway to network 90.0.0.0.

Consider the following example:

```
-> routeAdd "destination", "gate"
```

This call tells VxWorks that the machine with the host name “gate” is the gateway to the machine named “destination”. The host names “gate” and “destination” must already have been created by **hostAdd()**.

Consider the following example:

```
-> routeAdd "0", "gate"
```

This call tells VxWorks that the machine with the host name “gate” is the default gateway. The host “gate” must already have been created by **hostAdd()**. A default gateway is where Internet Protocol (IP) datagrams are routed when there is no specific routing table entry available for the destination IP network or host.

**RETURNS** OK or ERROR.

**SEE ALSO** routeLib

---

## routeDelete()

**NAME** routeDelete() – delete a route

**SYNOPSIS**

```
STATUS routeDelete
(
    char * destination,    /* inet addr or name of route destination */
    char * gateway        /* inet addr or name of gateway to destination */
)
```

**DESCRIPTION** This routine deletes a specified route from the network routing tables.

**RETURNS** OK or ERROR.

**SEE ALSO** routeLib, routeAdd()

---

## routeEntryAdd()

**NAME** routeEntryAdd() – insert a route in the routing table

**SYNOPSIS**

```
STATUS routeEntryAdd
(
    ROUTE_DESC * pRouteDesc /* information for new route entry */
)
```

**DESCRIPTION** This routine adds a route to the routing table. The *pRouteDesc* argument must include a destination address, gateway, and protocol identifier. If that argument does not include a netmask or specifies a netmask value of 0, the system creates a host-specific route entry. The *value1* through *value5*, and *routeTag* fields store arbitrary values for the new entry. The required *weight* field indicates the relative priority of the route (from 1 to 255) in case other entries to the same destination exist. The route with the lowest weight is visible to the IP forwarding process. A value of 0 will create an entry with the default weight value.

This routine ignores any values in the *flags*, *pIf*, and *pData* fields in the provided structure. If the add attempt is successful, the system sends callback messages and routing socket messages announcing the existence of the new route.

**RETURNS** OK on success and **ERROR** on failure.

**SEE ALSO** `routeEntryLib`

---

## routeEntryDel()

**NAME** `routeEntryDel()` – remove a route from the routing table

**SYNOPSIS**

```
STATUS routeEntryDel  
(  
    ROUTE_DESC * pRouteDesc    /* information for deleted route */  
)
```

**DESCRIPTION** This routine deletes a route in the routing table. The *pRouteDesc* argument must include a destination address. If that argument does not include a netmask or specifies a netmask value of 0, the system attempts to delete a host-specific route to the destination. If a route which matches the destination and netmask exists, a protocol ID of zero attempts to delete that entry (which is visible to the IP forwarding process) if the gateway value is not equal to zero. Otherwise, the system attempts to remove the first (lowest weight) entry which matches the provided protocol, or a specific entry within the first protocol group which also matches the supplied gateway address.

---

**NOTE:** This routine stores the actual gateway value in the *pRouteDesc* structure, so the corresponding buffer must be supplied even if no specific value is assigned. This routine does not use any fields in the *pRouteDesc* structure except the destination, gateway, netmask and protocol ID.

---

**RETURNS** OK on success, **ERROR** on failure

**SEE ALSO** `routeEntryLib`

---

## routeEntryLookup()

**NAME** routeEntryLookup() – find a matching route for a destination

**SYNOPSIS**

```
STATUS routeEntryLookup
(
    struct sockaddr * pDest, /* IP address reachable with matching route */
    ULONG *         pMask, /* netmask value, in network byte order */
    int             protoId, /* route source from m2Lib.h, or 0 for any. */
    ROUTE_DESC *   pRouteDesc /* information for matching route */
)
```

**DESCRIPTION** This routine searches the routing table for an entry which covers the specified destination address. It provides four types of searches based on the values of the *protoId* and *pMask* arguments.

If no mask is present (*pMask* is NULL) the search finds the matching entry with the longest netmask. Otherwise, the search ignores entries whose netmasks permit a match against the destination but do not equal the given value. Likewise, if *protoId* is not zero, the search restricts the possible matches to the specified route source.

Mask values of zero and 0xffffffff both indicate a host-specific route.

If neither value is specified, the search duplicates the results of the IP forwarding process for the destination (assuming no type-of-service match is required). It retrieves the matching entry with the longest netmask, regardless of the source which created it.

In all cases, if multiple entries match the search criteria, this routine selects the oldest one.

The chosen entry is copied into the supplied *pRouteDesc* structure, which is not modified if the search fails.

**RETURNS** OK if a route is found, or ERROR otherwise.

**SEE ALSO** routeEntryLib

---

## routeModify()

**NAME** routeModify() – change an entry in the routing table

**SYNOPSIS**

```
STATUS routeModify
(
    ROUTE_DESC *    pRouteDesc, /* information for matching route */
    struct sockaddr * pNewGateway /* new gateway, NULL if unchanged */
)
```

**DESCRIPTION** This routine searches the routing table for an entry which matches the destination address and netmask in the *pRouteDesc* structure. If the route descriptor structure does not include a netmask, it selects the longest netmask for the matching destination. A netmask value of zero searches for a host-specific route to the destination. A protocol ID of zero selects the first entry which matches the destination address and any netmask value. Otherwise, the search finds the route with the specified protocol. The retrieved route also matches any specified gateway value.

The *pNewGateway* argument supplies an optional replacement gateway address. The new address must be reachable through one of the local interfaces or the modification fails. The modification also fails if the destination address is not specified or if no route which matches the search criteria is found.

Once a route is chosen, this routine replaces the current metric values, route weight, and route tag with the corresponding entries in the *pRouteDesc* structure. The pointers for the interface and additional data in the *pRouteDesc* argument are not used. The route flags are also not changed.

---

**NOTE:** Changing the weight of a route will reorganize any duplicate routes and may alter which entry is visible to the IP forwarding process.

---

**RETURNS** OK if a route is found and changed, or **ERROR** otherwise.

**SEE ALSO** routeEntryLib

---

## routeNetAdd()

**NAME** `routeNetAdd()` – add a route to a destination that is a network

**SYNOPSIS**

```
STATUS routeNetAdd
(
    char * destination,    /* inet addr or name of network destination */
    char * gateway        /* inet addr or name of gateway to destination */
)
```

**DESCRIPTION** This routine is equivalent to `routeAdd()`, except that the destination address is assumed to be a network. This is useful for adding a route to a sub-network that is not on the same overall network as the local network.

This routine can be used to add multiple routes to the same destination differing by the gateway.

**RETURNS** OK or ERROR.

**SEE ALSO** `routeLib`

---

## routeShow()

**NAME** `routeShow()` – display all IP routes (summary information)

**SYNOPSIS**

```
void routeShow (void)
```

**DESCRIPTION** This routine displays the list of destinations in the routing table along with the next-hop gateway and associated interface for each entry. It separates the routes into network routes and host-specific entries, but does not display the netmask for a route since it was created for class-based routes which used predetermined values for that field.

The IP forwarding process will only use the first route entry to a destination. When multiple routes exist to the same destination with the same netmask (which is not shown), the first route entry uses the lowest administrative weight. The remaining entries (listed as additional routes) use the same destination address. One of those entries will replace the primary route if it is deleted.

**EXAMPLE**

```
-> routeShow
ROUTE NET TABLE
Destination      Gateway          Flags  Refcnt Use      Interface
```

```

-----
90.0.0.0      90.0.0.63    0x1  1    142    enp0
10.0.0.0     90.0.0.70    0x1  1    142    enp0
  Additional routes to 10.0.0.0:
            80.0.0.70    0x1  0    120    enp1
-----
ROUTE HOST TABLE
Destination    Gateway        Flags  Refcnt  Use      Interface
-----
127.0.0.1     127.0.0.1     0x101  0       82      lo0
-----

```

The flags field represents a decimal value of the flags specified for a given route. The following is a list of currently available flag values:

- 0x1 - route is usable (that is, “up”)
- 0x2 - destination is a gateway
- 0x4 - host specific routing entry
- 0x8 - host or net unreachable
- 0x10 - created dynamically (by redirect)
- 0x20 - modified dynamically (by redirect)
- 0x40 - message confirmed
- 0x80 - subnet mask present
- 0x100 - generate new routes on use
- 0x200 - external daemon resolves name
- 0x400 - generated by ARP
- 0x800 - manually added (static)
- 0x1000 - just discard packets (during updates)
- 0x2000 - modified by management protocol
- 0x4000 - protocol specific routing flag
- 0x8000 - protocol specific routing flag

In the above display example, the entry in the ROUTE NET TABLE has a flag value of 1, which indicates that this route is “up” and usable and network specific (the 0x4 bit is turned off). The entry in the ROUTE HOST TABLE has a flag value of 5 (0x1 OR’ed with 0x4), which indicates that this route is “up” and usable and host-specific.

Some configuration is required when this routine is to be used remotely over the network, *e.g.*, through a **telnet** session or through the host shell using **WDB\_COMM\_NETWORK**. If, more than 5 routes are expected in the table the parameter **RT\_BUFFERED\_DISPLAY** should be set to **TRUE** to prevent a possible deadlock. This requires a buffer whose size can be set with **RT\_DISPLAY\_MEMORY**. It will limit the number of routes that can be displayed (each route requires approx. 70 bytes).

**RETURNS** N/A

**SEE ALSO** netShow

**R**

## **routeStatShow()**

<b>NAME</b>	<b>routeStatShow()</b> – display routing statistics
<b>SYNOPSIS</b>	<code>void routeStatShow (void)</code>
<b>DESCRIPTION</b>	This routine displays routing statistics.
<b>RETURNS</b>	N/A
<b>SEE ALSO</b>	<b>netShow</b>

---

## **routeStorageUnbind()**

<b>NAME</b>	<b>routeStorageUnbind()</b> – remove a registered handler from the routing system
<b>SYNOPSIS</b>	<pre><b>STATUS</b> routeStorageUnbind (     void * pCookie          /* identifier from routeStorageBind() routine */ )</pre>
<b>DESCRIPTION</b>	A routing protocol uses this routine to prevent a registered function from receiving any callback messages. Any data accessible with the extra argument to that function must be maintained until this routine completes successfully.
<b>RETURNS</b>	OK if removal succeeds, or <b>ERROR</b> otherwise.
<b>SEE ALSO</b>	<b>routeMessageLib</b>

---

## routeTableWalk()

**NAME** `routeTableWalk()` – traverse the IP routing table

**SYNOPSIS**

```
STATUS routeTableWalk
(
    struct sockaddr * pDest, /* destination address, or NULL if none. */
    int             protoId, /* route source, or 0 for any. */
    VOIDFUNCPTR    pFunc,   /* callback function */
    void *          pArg    /* optional callback function argument */
)
```

**DESCRIPTION** This routine applies the provided function to every entry in the IP routing table which meets the criteria indicated by the *pDest* and *protoId* arguments. If a destination address is specified, the given function executes for each route table entry which matches the destination. If a protocol identifier is supplied, the function executes for each entry created by the protocol instead. If no value is specified, the routine displays every entry in the table. The supplied argument *pArg* is passed back to callback function.

**RETURNS** OK if traversal completes, or **ERROR** otherwise.

---

**NOTE:** Only one of the two values *pDest* and *protoId* should be specified. Specifying both results in **ERROR** being returned.

---

**NOTE:** The provided routine executes while the system holds internal locks which restrict all network stack activity and any routing operations to the calling task. That routine **MUST NOT** perform any operations which alter the existing routing table. This walk routine relies on a fixed order of all route entries to complete. Creating or removing route entries could corrupt the table, causing the calling task to enter an endless loop or halt completely. That behavior would deadlock the entire network system, since other tasks would wait indefinitely for the unavailable locks.

---

**SEE ALSO** `routeEntryLib`

## rpcInit()

<b>NAME</b>	<b>rpcInit()</b> – initialize the RPC package
<b>SYNOPSIS</b>	<b>STATUS</b> <b>rpcInit</b> ( <b>void</b> )
<b>DESCRIPTION</b>	This routine must be called before any task can use the RPC facility; it spawns the <b>portmap</b> daemon. It is called automatically if <b>INCLUDE_RPC</b> is defined.
<b>VXWORKS AE PROTECTION DOMAINS</b>	Under VxWorks AE, you can call this function from within the kernel protection domain only. This restriction does not apply under non-AE versions of VxWorks.
<b>RETURNS</b>	<b>OK</b> , or <b>ERROR</b> if the portmap daemon cannot be spawned.
<b>SEE ALSO</b>	<b>rpcLib</b>

---

## rpcTaskInit()

<b>NAME</b>	<b>rpcTaskInit()</b> – initialize a task's access to the RPC package
<b>SYNOPSIS</b>	<b>STATUS</b> <b>rpcTaskInit</b> ( <b>void</b> )
<b>DESCRIPTION</b>	This routine must be called by a task before it makes any calls to other routines in the RPC package.
<b>VXWORKS AE PROTECTION DOMAINS</b>	Under VxWorks AE, you can call this function from within the kernel protection domain only. This restriction does not apply under non-AE versions of VxWorks.
<b>RETURNS</b>	<b>OK</b> , or <b>ERROR</b> if there is insufficient memory or the routine is unable to add a task delete hook.
<b>SEE ALSO</b>	<b>rpcLib</b>

---

## rresvport( )

- NAME** `rresvport( )` – open a socket with a privileged port bound to it
- SYNOPSIS**
- ```
int rresvport
(
    int * alport          /* port number to initially try */
)
```
- DESCRIPTION** This routine opens a socket with a privileged port bound to it. It is analogous to the UNIX routine `rresvport( )`.
- RETURNS** A socket descriptor, or `ERROR` if either the socket cannot be opened or all ports are in use.
- SEE ALSO** `remLib`, UNIX BSD 4.3 manual entry for `rresvport( )`

---

## rt11FsDateSet( )

- NAME** `rt11FsDateSet( )` – set the rt11Fs file system date
- SYNOPSIS**
- ```
void rt11FsDateSet
(
    int year,            /* year (72...03 (RT-11's days are numbered)) */
    int month,          /* month (0, or 1...12) */
    int day              /* day (0, or 1...31) */
)
```
- DESCRIPTION** This routine sets the date for the rt11Fs file system, which remains in effect until changed. All files created are assigned this creation date.
- To set a blank date, invoke the command:
- ```
rt11FsDateSet (72, 0, 0); /* a date outside RT-11's epoch */
```
- NOTE:** No automatic incrementing of the date is performed; each new date must be set with a call to this routine.
- 
- RETURNS** N/A
- SEE ALSO** `rt11FsLib`

---

## rt11FsDevInit()

**NAME**            **rt11FsDevInit()** – initialize the rt11Fs device descriptor

**SYNOPSIS**        **RT\_VOL\_DESC \*rt11FsDevInit**  
                  (  
                  **char \*     devName,**        **/\* device name \*/**  
                  **BLK\_DEV \* pBlkDev,**    **/\* pointer to block device info \*/**  
                  **BOOL     rt11Fmt,**     **/\* TRUE if RT-11 skew & interleave \*/**  
                  **int     nEntries,**    **/\* no. of dir entries incl term entry \*/**  
                  **BOOL     changeNoWarn** **/\* TRUE if no disk change warning \*/**  
                  )

**DESCRIPTION**    This routine initializes the device descriptor. The *pBlkDev* parameter is a pointer to an already-created **BLK\_DEV** device structure. This structure contains definitions for various aspects of the physical device format, as well as pointers to the sector read, sector write, **ioctl()**, status check, and reset functions for the device.

The *rt11Fmt* parameter is **TRUE** if the device is to be accessed using standard RT-11 skew and interleave.

The device directory will consist of one segment able to contain at least as many files as specified by *nEntries*. If *nEntries* is equal to **RT\_FILES\_FOR\_2\_BLOCK\_SEG**, strict RT-11 compatibility is maintained.

The *changeNoWarn* parameter is **TRUE** if the disk may be changed without announcing the change via **rt11FsReadyChange()**. Setting *changeNoWarn* to **TRUE** causes the disk to be regularly remounted, in case it has been changed. This results in a significant performance penalty.

---

**NOTE:** An **ERROR** is returned if *rt11Fmt* is **TRUE** and the **bd\_blksPerTrack** (sectors per track) field in the **BLK\_DEV** structure is odd. This is because an odd number of sectors per track is incompatible with the RT-11 interleaving algorithm.

---

**RETURNS**        A pointer to the volume descriptor (**RT\_VOL\_DESC**), or **NULL** if invalid device parameters were specified, or the routine runs out of memory.

**SEE ALSO**        **rt11FsLib**

---

## rt11FsInit( )

|                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
|--------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>        | rt11FsInit( ) – prepare to use the rt11Fs library                                                                                                                                                                                                                                                                                                                                                                                                                          |
| <b>SYNOPSIS</b>    | <pre> <b>STATUS</b> rt11FsInit     (         <b>int</b> maxFiles           /* max no. of simultaneously open rt11Fs files */     ) </pre>                                                                                                                                                                                                                                                                                                                                  |
| <b>DESCRIPTION</b> | <p>This routine initializes the rt11Fs library. It must be called exactly once, before any other routine in the library. The <i>maxFiles</i> parameter specifies the number of rt11Fs files that may be open at once. This routine initializes the necessary memory structures and semaphores.</p> <p>This routine is called automatically from the root task, <b>usrRoot( )</b>, in <b>usrConfig.c</b> when the configuration macro <b>INCLUDE_RT11FS</b> is defined.</p> |
| <b>RETURNS</b>     | OK, or <b>ERROR</b> if memory is insufficient.                                                                                                                                                                                                                                                                                                                                                                                                                             |
| <b>SEE ALSO</b>    | rt11FsLib                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |

---

## rt11FsMkfs( )

|                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
|--------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>        | rt11FsMkfs( ) – initialize a device and create an rt11Fs file system                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| <b>SYNOPSIS</b>    | <pre> <b>RT_VOL_DESC</b> *rt11FsMkfs     (         <b>char</b> *   volName,           /* volume name to use */         <b>BLK_DEV</b> * pBlkDev          /* pointer to block device struct */     ) </pre>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
| <b>DESCRIPTION</b> | <p>This routine provides a quick method of creating an rt11Fs file system on a device. It is used instead of the two-step procedure of calling <b>rt11FsDevInit( )</b> followed by an <b>ioctl( )</b> call with an <b>FIODISKINIT</b> function code.</p> <p>This routine provides defaults for the rt11Fs parameters expected by <b>rt11FsDevInit( )</b>. The directory size is set to <b>RT_FILES_FOR_2_BLOCK_SEG</b> (defined in <b>rt11FsLib.h</b>). No standard disk format is assumed; this allows the use of rt11Fs on block devices with an odd number of sectors per track. The <i>changeNoWarn</i> parameter is defined as <b>FALSE</b>, indicating that the disk will not be replaced without <b>rt11FsReadyChange( )</b> being called first.</p> |

If different values are needed for any of these parameters, the routine **rt11FsDevInit()** must be used instead of this routine, followed by a request for disk initialization using the **ioctl()** function **FIODISKINIT**.

**RETURNS** A pointer to an rt11Fs volume descriptor (**RT\_VOL\_DESC**), or **NULL** if there is an error.

**SEE ALSO** **rt11FsLib**, **rt11FsDevInit()**

---

## rt11FsModeChange()

**NAME** **rt11FsModeChange()** – modify the mode of an rt11Fs volume

**SYNOPSIS**

```
void rt11FsModeChange
(
    RT_VOL_DESC * vdptr,      /* pointer to volume descriptor */
    int          newMode      /* O_RDONLY, O_WRONLY, or O_RDWR (both) */
)
```

**DESCRIPTION** This routine sets the volume descriptor mode to *newMode*. It should be called whenever the read and write capabilities are determined, usually after a ready change. See the manual entry for **rt11FsReadyChange()**.

The **rt11FsDevInit()** routine initially sets the mode to **O\_RDWR**, (e.g., both **O\_RDONLY** and **O\_WRONLY**).

**RETURNS** N/A

**SEE ALSO** **rt11FsLib**, **rt11FsDevInit()**, **rt11FsReadyChange()**

---

## rt11FsReadyChange()

**NAME** **rt11FsReadyChange()** – notify rt11Fs of a change in ready status

**SYNOPSIS**

```
void rt11FsReadyChange
(
    RT_VOL_DESC * vdptr      /* pointer to device descriptor */
)
```

|                    |                                                                                                                                                                                                                                  |
|--------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>DESCRIPTION</b> | This routine sets the volume descriptor state to <b>RT_VD_READY_CHANGED</b> . It should be called whenever a driver senses that a device has come on-line or gone off-line ( <i>e.g.</i> , a disk has been inserted or removed). |
| <b>RETURNS</b>     | N/A                                                                                                                                                                                                                              |
| <b>SEE ALSO</b>    | <b>rt11FsLib</b>                                                                                                                                                                                                                 |

**s()**


---

|                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
|--------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>        | <b>s()</b> – single-step a task                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| <b>SYNOPSIS</b>    | <pre> <b>STATUS s</b> (   int      taskNameOrId,      /* task to step; 0 = use default */   INSTR * addr,              /* address to step to; 0 = next instruction */ ) </pre>                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
| <b>DESCRIPTION</b> | <p>This routine single-steps a task that is stopped at a breakpoint.</p> <p>To execute, enter:</p> <pre>-&gt; s [task[,addr[,addr1]]]</pre> <p>If <i>task</i> is omitted or zero, the last task referenced is assumed. If <i>addr</i> is non-zero, then the program counter is changed to <i>addr</i>; if <i>addr1</i> is non-zero, the next program counter is changed to <i>addr1</i>, and the task is stepped.</p> <hr/> <p><b>WARNING:</b> When a task is continued, <b>s()</b> does not distinguish between a suspended task or a task suspended by the debugger. Therefore, its use should be restricted to only those tasks being debugged.</p> <hr/> |
| <b>RETURNS</b>     | <b>OK</b> , or <b>ERROR</b> if the debugging package is not installed, the task cannot be found, or the task is not suspended.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
| <b>SEE ALSO</b>    | <b>dbgLib</b> , <i>VxWorks Programmer's Guide: Target Shell</i> , <b>windsh</b> , <i>Tornado User's Guide: Shell</i>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |

---

**scanf()**

|                    |                                                                                                                                                                                            |
|--------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>        | <b>scanf()</b> – read and convert characters from the standard input stream (ANSI)                                                                                                         |
| <b>SYNOPSIS</b>    | <pre> int scanf (   char const * fmt,          /* format string */   ...                       /* arguments to format string */ ) </pre>                                                   |
| <b>DESCRIPTION</b> | <p>This routine reads input from the standard input stream under the control of the string <i>fmt</i>. It is equivalent to <b>fscanf()</b> with an <i>fp</i> argument of <b>stdin</b>.</p> |

|                      |                                                                                                                                                                                             |
|----------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>INCLUDE FILES</b> | <code>stdio.h</code>                                                                                                                                                                        |
| <b>RETURNS</b>       | The number of input items assigned, which can be fewer than provided for, or even zero, in the event of an early matching failure; or EOF if an input failure occurs before any conversion. |
| <b>SEE ALSO</b>      | <code>ansiStdio</code> , <code>fscanf()</code> , <code>sscanf()</code>                                                                                                                      |

---

## ***sched\_get\_priority\_max()***

**NAME** `sched_get_priority_max()` – get the maximum priority (POSIX)

**SYNOPSIS**

```
int sched_get_priority_max  
(  
    int policy                /* scheduling policy */  
)
```

**DESCRIPTION** This routine returns the value of the highest possible task priority for a specified scheduling policy (`SCHED_FIFO` or `SCHED_RR`).

---

**NOTE:** If the global variable `posixPriorityNumbering` is `FALSE`, the VxWorks native priority numbering scheme is used, in which higher priorities are indicated by smaller numbers. This is different than the priority numbering scheme specified by POSIX, in which higher priorities are indicated by larger numbers.

---

**RETURNS** Maximum priority value, or -1 (**ERROR**) on error.

**ERRNO** `EINVAL`  
- invalid scheduling policy.

**SEE ALSO** `schedPxLib`

---

## **sched\_get\_priority\_min()**

**NAME** `sched_get_priority_min()` – get the minimum priority (POSIX)

**SYNOPSIS**

```
int sched_get_priority_min
(
    int policy          /* scheduling policy */
)
```

**DESCRIPTION** This routine returns the value of the lowest possible task priority for a specified scheduling policy (`SCHED_FIFO` or `SCHED_RR`).

---

**NOTE:** If the global variable `posixPriorityNumbering` is `FALSE`, the VxWorks native priority numbering scheme is used, in which higher priorities are indicated by smaller numbers. This is different than the priority numbering scheme specified by POSIX, in which higher priorities are indicated by larger numbers.

---

**RETURNS** Minimum priority value, or -1 (`ERROR`) on error.

**ERRNO** `EINVAL`  
- invalid scheduling policy.

**SEE ALSO** `schedPxBLib`

---

## **sched\_getparam()**

**NAME** `sched_getparam()` – get the scheduling parameters for a specified task (POSIX)

**SYNOPSIS**

```
int sched_getparam
(
    pid_t      tid, /* task ID */
    struct sched_param * param /* scheduling param to store priority */
)
```

**DESCRIPTION** This routine gets the scheduling priority for a specified task, *tid*. If *tid* is 0, it gets the priority of the calling task. The task's priority is copied to the `sched_param` structure pointed to by *param*.

---

**NOTE:** If the global variable **posixPriorityNumbering** is **FALSE**, the VxWorks native priority numbering scheme is used, in which higher priorities are indicated by smaller numbers. This is different than the priority numbering scheme specified by POSIX, in which higher priorities are indicated by larger numbers.

---

**RETURNS** 0 (OK) if successful, or -1 (ERROR) on error.

**ERRNO** ESRCH  
- invalid task ID.

**SEE ALSO** schedPxLib

---

## sched\_getscheduler()

**NAME** sched\_getscheduler() – get the current scheduling policy (POSIX)

**SYNOPSIS**

```
int sched_getscheduler
(
    pid_t tid          /* task ID */
)
```

**DESCRIPTION** This routine returns the current scheduling policy (*i.e.*, SCHED\_FIFO or SCHED\_RR).

**RETURNS** Current scheduling policy (SCHED\_FIFO or SCHED\_RR), or -1 (ERROR) on error.

**ERRNO** ESRCH  
- invalid task ID.

**SEE ALSO** schedPxLib

---

## **sched\_rr\_get\_interval()**

|                    |                                                                                                                                                    |
|--------------------|----------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>        | <b>sched_rr_get_interval()</b> – get the current time slice (POSIX)                                                                                |
| <b>SYNOPSIS</b>    | <pre>int sched_rr_get_interval (     pid_t          tid,    /* task ID */     struct timespec * interval /* struct to store time slice */ ) </pre> |
| <b>DESCRIPTION</b> | This routine sets <i>interval</i> to the current time slice period if round-robin scheduling is currently enabled.                                 |
| <b>RETURNS</b>     | 0 (OK) if successful, -1 (ERROR) on error.                                                                                                         |
| <b>ERRNO</b>       | <p>EINVAL<br/>- round-robin scheduling is not currently enabled.</p> <p>ESRCH<br/>- invalid task ID.</p>                                           |
| <b>SEE ALSO</b>    | <b>schedPxBLib</b>                                                                                                                                 |

---

## **sched\_setparam()**

|                    |                                                                                                                                                                                                                                                                                                                                                                                                                                               |
|--------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>        | <b>sched_setparam()</b> – set a task's priority (POSIX)                                                                                                                                                                                                                                                                                                                                                                                       |
| <b>SYNOPSIS</b>    | <pre>int sched_setparam (     pid_t          tid,    /* task ID */     const struct sched_param * param /* scheduling parameter */ ) </pre>                                                                                                                                                                                                                                                                                                   |
| <b>DESCRIPTION</b> | <p>This routine sets the priority of a specified task, <i>tid</i>. If <i>tid</i> is 0, it sets the priority of the calling task. Valid priority numbers are 0 through 255.</p> <p>The <i>param</i> argument is a structure whose member <b>sched_priority</b> is the integer priority value. For example, the following program fragment sets the calling task's priority to 13 using POSIX interfaces:</p> <pre>#include "sched.h" ...</pre> |

```

struct sched_param AppSchedPrio;
...
AppSchedPrio.sched_priority = 13;
if ( sched_setparam (0, &AppSchedPrio) != OK )
    {
        ... /* recovery attempt or abort message */
    }
...

```

---

**NOTE:** If the global variable `posixPriorityNumbering` is `FALSE`, the VxWorks native priority numbering scheme is used, in which higher priorities are indicated by smaller numbers. This is different than the priority numbering scheme specified by POSIX, in which higher priorities are indicated by larger numbers.

---

**RETURNS** 0 (OK) if successful, or -1 (ERROR) on error.

**ERRNO** EINVAL  
 - scheduling priority is outside valid range.  
 ESRCH  
 - task ID is invalid.

**SEE ALSO** `schedPxBLib`

---

## sched\_setscheduler()

**NAME** `sched_setscheduler()` – set scheduling policy and scheduling parameters (POSIX)

**SYNOPSIS**

```

int sched_setscheduler
(
    pid_t          tid, /* task ID */
    int            policy, /* scheduling policy requested */
    const struct sched_param * param /* scheduling parameters requested */
)

```

**DESCRIPTION** This routine sets the scheduling policy and scheduling parameters for a specified task, *tid*. If *tid* is 0, it sets the scheduling policy and scheduling parameters for the calling task.

Because VxWorks does not set scheduling policies (e.g., round-robin scheduling) on a task-by-task basis, setting a scheduling policy that conflicts with the current system policy simply fails and `errno` is set to `EINVAL`. If the requested scheduling policy is the same as the current system policy, then this routine acts just like `sched_setparam()`.



## ***sched\_yield()***

---

**NOTE:** If the global variable **posixPriorityNumbering** is **FALSE**, the VxWorks native priority numbering scheme is used, in which higher priorities are indicated by smaller numbers. This is different than the priority numbering scheme specified by POSIX, in which higher priorities are indicated by larger numbers.

---

**RETURNS** The previous scheduling policy (**SCHED\_FIFO** or **SCHED\_RR**), or -1 (**ERROR**) on error.

**ERRNO** **EINVAL**  
- scheduling priority is outside valid range, or it is impossible to set the specified scheduling policy.  
**ESRCH**  
- invalid task ID.

**SEE ALSO** **schedPxLib**

---

## **sched\_yield()**

**NAME** **sched\_yield()** – relinquish the CPU (POSIX)

**SYNOPSIS** **int sched\_yield (void)**

**DESCRIPTION** This routine forces the running task to give up the CPU.

**RETURNS** 0 (**OK**) if successful, or -1 (**ERROR**) on error.

**SEE ALSO** **schedPxLib**

---

## **scsi2IfInit()**

**NAME** **scsi2IfInit()** – initialize the SCSI-2 interface to **scsiLib**

**SYNOPSIS** **void scsi2IfInit ()**

**DESCRIPTION** This routine initializes the SCSI-2 function interface by adding all the routines in **scsi2Lib** plus those in **scsiDirectLib** and **scsiCommonLib**. It is invoked by **usrConfig.c** if the macro **INCLUDE\_SCSI2** is defined in **config.h**. The calling interface remains the same

between SCSI-1 and SCSI-2; this routine simply sets the calling interface function pointers to the SCSI-2 functions.

**RETURNS** N/A

**SEE ALSO** `scsi2Lib`

---

## scsiAutoConfig()

**NAME** `scsiAutoConfig()` – configure all devices connected to a SCSI controller

**SYNOPSIS**

```
STATUS scsiAutoConfig
(
    SCSI_CTRL * pScsiCtrl    /* ptr to SCSI controller info */
)
```

**DESCRIPTION** This routine cycles through all valid SCSI bus IDs and logical unit numbers (LUNs), attempting a `scsiPhysDevCreate()` with default parameters on each. All devices which support the INQUIRY command are configured. The `scsiShow()` routine can be used to find the system table of SCSI physical devices attached to a specified SCSI controller. In addition, `scsiPhysDevIdGet()` can be used programmatically to get a pointer to the `SCSI_PHYS_DEV` structure associated with the device at a specified SCSI bus ID and LUN.

**RETURNS** OK, or ERROR if `pScsiCtrl` and the global variable `pSysScsiCtrl` are both NULL.

**SEE ALSO** `scsiLib`

---

## scsiBlkDevCreate()

**NAME** `scsiBlkDevCreate()` – define a logical partition on a SCSI block device

**SYNOPSIS**

```
BLK_DEV * scsiBlkDevCreate
(
    SCSI_PHYS_DEV * pScsiPhysDev, /* ptr to SCSI physical device info */
    int             numBlocks,    /* number of blocks in block device */
    int             blockOffset   /* address of first block in volume */
)
```

**scsiBlkDevInit()**

**DESCRIPTION** This routine creates and initializes a **BLK\_DEV** structure, which describes a logical partition on a SCSI physical-block device. A logical partition is an array of contiguously addressed blocks; it can be completely described by the number of blocks and the address of the first block in the partition. In normal configurations partitions do not overlap, although such a condition is not an error.

---

**NOTE:** If *numBlocks* is 0, the rest of device is used.

---

**RETURNS** A pointer to the created **BLK\_DEV**, or **NULL** if parameters exceed physical device boundaries, if the physical device is not a block device, or if memory is insufficient for the structures.

**SEE ALSO** **scsiLib**

---

## scsiBlkDevInit()

**NAME** **scsiBlkDevInit()** – initialize fields in a SCSI logical partition

**SYNOPSIS**

```
void scsiBlkDevInit
(
    SCSI_BLK_DEV * pScsiBlkDev, /* ptr to SCSI block dev. struct */
    int           blksPerTrack, /* blocks per track */
    int           nHeads       /* number of heads */
)
```

**DESCRIPTION** This routine specifies the disk-geometry parameters required by certain file systems (for example, dosFs). It is called after a **SCSI\_BLK\_DEV** structure is created with **scsiBlkDevCreate()**, but before calling a file system initialization routine. It is generally required only for removable-media devices.

**RETURNS** N/A

**SEE ALSO** **scsiLib**

---

## scsiBlkDevShow()

|                    |                                                                                                                                                                                                                      |
|--------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>        | <code>scsiBlkDevShow()</code> – show the <code>BLK_DEV</code> structures on a specified physical device                                                                                                              |
| <b>SYNOPSIS</b>    | <pre>void scsiBlkDevShow (     SCSI_PHYS_DEV * pScsiPhysDev /* ptr to SCSI physical device info */ )</pre>                                                                                                           |
| <b>DESCRIPTION</b> | This routine displays all of the <code>BLK_DEV</code> structures created on a specified physical device. This routine is called by <code>scsiShow()</code> but may also be invoked directly, usually from the shell. |
| <b>RETURNS</b>     | N/A                                                                                                                                                                                                                  |
| <b>SEE ALSO</b>    | <code>scsiLib</code> , <code>scsiShow()</code>                                                                                                                                                                       |

---

## scsiBusReset()

|                    |                                                                                                                                                                                                                                |
|--------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>        | <code>scsiBusReset()</code> – pulse the reset signal on the SCSI bus                                                                                                                                                           |
| <b>SYNOPSIS</b>    | <pre>STATUS scsiBusReset (     SCSI_CTRL * pScsiCtrl /* ptr to SCSI controller info */ )</pre>                                                                                                                                 |
| <b>DESCRIPTION</b> | This routine calls a controller-specific routine to reset a specified controller's SCSI bus. If no controller is specified ( <code>pScsiCtrl</code> is 0), the value in the global variable <code>pSysScsiCtrl</code> is used. |
| <b>RETURNS</b>     | OK, or ERROR if there is no controller or controller-specific routine.                                                                                                                                                         |
| <b>SEE ALSO</b>    | <code>scsiLib</code>                                                                                                                                                                                                           |

---

## scsiCacheSnoopDisable()

**NAME** `scsiCacheSnoopDisable()` – inform SCSI that hardware snooping of caches is disabled

**SYNOPSIS**

```
void scsiCacheSnoopDisable
(
    SCSI_CTRL * pScsiCtrl    /* pointer to a SCSI_CTRL structure */
)
```

**DESCRIPTION** This routine informs the SCSI library that hardware snooping is disabled and that **scsi2Lib** should execute any necessary cache coherency code. In order to make **scsi2Lib** aware that hardware snooping is disabled, this routine should be called after all SCSI-2 initializations, especially after **scsi2CtrlInit()**.

**RETURNS** N/A

**SEE ALSO** `scsi2Lib`

---

## scsiCacheSnoopEnable()

**NAME** `scsiCacheSnoopEnable()` – inform SCSI that hardware snooping of caches is enabled

**SYNOPSIS**

```
void scsiCacheSnoopEnable
(
    SCSI_CTRL * pScsiCtrl    /* pointer to a SCSI_CTRL structure */
)
```

**DESCRIPTION** This routine informs the SCSI library that hardware snooping is enabled and that **scsi2Lib** need not execute any cache coherency code. In order to make **scsi2Lib** aware that hardware snooping is enabled, this routine should be called after all SCSI-2 initializations, especially after **scsi2CtrlInit()**.

**RETURNS** N/A

**SEE ALSO** `scsi2Lib`

---

## scsiCacheSynchronize()

**NAME** `scsiCacheSynchronize()` – synchronize the caches for data coherency

**SYNOPSIS**

```
void scsiCacheSynchronize
(
    SCSI_THREAD *    pThread, /* ptr to thread info */
    SCSI_CACHE_ACTION action /* cache action required */
)
```

**DESCRIPTION** This routine performs whatever cache action is necessary to ensure cache coherency with respect to the various buffers involved in a SCSI command. The process is as follows:

1. The buffers for command, identification, and write data, which are simply written to SCSI, are flushed before the command.
2. The status buffer, which is written and then read, is cleared (flushed and invalidated) before the command.
3. The data buffer for a read command, which is only read, is cleared before the command.

The data buffer for a read command is cleared before the command rather than invalidated after it because it may share dirty cache lines with data outside the read buffer. DMA drivers for older versions of the SCSI library have flushed the first and last bytes of the data buffer before the command. However, this approach is not sufficient with the enhanced SCSI library because the amount of data transferred into the buffer may not fill it, which would cause dirty cache lines which contain correct data for the un-filled part of the buffer to be lost when the buffer is invalidated after the command.

To optimize the performance of the driver in supporting different caching policies, the routine uses the `CACHE_USER_FLUSH` macro when flushing the cache. In the absence of a `CACHE_USER_CLEAR` macro, the following steps are taken:

1. If there is a non-NULL flush routine in the `cacheUserFuncs` structure, the cache is cleared.
2. If there is a non-NULL invalidate routine, the cache is invalidated.
3. Otherwise nothing is done; the cache is assumed to be coherent without any software intervention.

Finally, since flushing (clearing) cache line entries for a large data buffer can be time-consuming, if the data buffer is larger than a preset (run-time configurable) size, the entire cache is flushed.

**RETURNS** N/A

**SEE ALSO** `scsi2Lib`

## scsiErase()

**NAME** scsiErase() – issue an ERASE command to a SCSI device

**SYNOPSIS**

```
STATUS scsiErase
(
    SCSI_PHYS_DEV * pScsiPhysDev, /* ptr to SCSI physical device */
    BOOL          longErase      /* TRUE for entire tape erase */
)
```

**DESCRIPTION** This routine issues an ERASE command to a specified SCSI device.

**RETURNS** OK, or ERROR if the command fails.

**SEE ALSO** scsiSeqLib

---

## scsiFormatUnit()

**NAME** scsiFormatUnit() – issue a FORMAT\_UNIT command to a SCSI device

**SYNOPSIS**

```
STATUS scsiFormatUnit
(
    SCSI_PHYS_DEV * pScsiPhysDev, /* ptr to SCSI physical device */
    BOOL          cmpDefectList, /* whether defect list is complete */
    int           defListFormat, /* defect list format */
    int           vendorUnique, /* vendor unique byte */
    int           interleave, /* interleave factor */
    char *        buffer, /* ptr to input data buffer */
    int           bufLength /* length of buffer in bytes */
)
```

**DESCRIPTION** This routine issues a FORMAT\_UNIT command to a specified SCSI device.

**RETURNS** OK, or ERROR if the command fails.

**SEE ALSO** scsiLib

---

## scsiIdentMsgBuild()

**NAME** `scsiIdentMsgBuild()` – build an identification message

**SYNOPSIS**

```
int scsiIdentMsgBuild
(
    UINT8 *      msg,
    SCSI_PHYS_DEV * pScsiPhysDev,
    SCSI_TAG_TYPE tagType,
    UINT         tagNumber
)
```

**DESCRIPTION** This routine builds an identification message in the caller's buffer, based on the specified physical device, tag type, and tag number.

If the target device does not support messages, there is no identification message to build.

Otherwise, the identification message consists of an IDENTIFY byte plus an optional QUEUE TAG message (two bytes), depending on the type of tag used.

---

**NOTE:** This function is not intended for use by application programs.

---

**RETURNS** The length of the resulting identification message in bytes or -1 for **ERROR**.

**SEE ALSO** `scsi2Lib`

---

## scsiIdentMsgParse()

**NAME** `scsiIdentMsgParse()` – parse an identification message

**SYNOPSIS**

```
SCSI_IDENT_STATUS scsiIdentMsgParse
(
    SCSI_CTRL *      pScsiCtrl,
    UINT8 *         msg,
    int             msgLength,
    SCSI_PHYS_DEV * * ppScsiPhysDev,
    SCSI_TAG *      pTagNum
)
```

**scsiInquiry()**

|                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
|--------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>DESCRIPTION</b> | <p>This routine scans a (possibly incomplete) identification message, validating it in the process. If there is an IDENTIFY message, it identifies the corresponding physical device.</p> <p>If the physical device is currently processing an untagged (ITL) nexus, identification is complete. Otherwise, the identification is complete only if there is a complete QUEUE TAG message.</p> <p>If there is no physical device corresponding to the IDENTIFY message, or if the device is processing tagged (ITLQ) nexuses and the tag does not correspond to an active thread (it may have been aborted by a timeout, for example), then the identification sequence fails.</p> <p>The caller's buffers for physical device and tag number (the results of the identification process) are always updated. This is required by the thread event handler (see <b>scsiMgrThreadEvent()</b>.)</p> <hr/> <p><b>NOTE:</b> This function is not intended for use by application programs.</p> <hr/> |
| <b>RETURNS</b>     | The identification status (incomplete, complete, or rejected).                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
| <b>SEE ALSO</b>    | <b>scsi2Lib</b>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |

---

## scsiInquiry()

|                    |                                                                                                                                                                                                                                                     |
|--------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>        | <b>scsiInquiry()</b> – issue an INQUIRY command to a SCSI device                                                                                                                                                                                    |
| <b>SYNOPSIS</b>    | <pre> <b>STATUS</b> scsiInquiry (     SCSI_PHYS_DEV * pScsiPhysDev, /* ptr to SCSI physical device */     char *          buffer,        /* ptr to input data buffer */     int             bufLength      /* length of buffer in bytes */ ) </pre> |
| <b>DESCRIPTION</b> | This routine issues an INQUIRY command to a specified SCSI device.                                                                                                                                                                                  |
| <b>RETURNS</b>     | OK, or ERROR if the command fails.                                                                                                                                                                                                                  |
| <b>SEE ALSO</b>    | <b>scsiLib</b>                                                                                                                                                                                                                                      |

---

## scsiIoctl()

**NAME** `scsiIoctl()` – perform a device-specific I/O control function

**SYNOPSIS**

```
STATUS scsiIoctl
(
    SCSI_PHYS_DEV * pScsiPhysDev, /* ptr to SCSI block device info */
    int            function,      /* function code */
    int            arg            /* argument to pass called function */
)
```

**DESCRIPTION** This routine performs a specified `ioctl` function using a specified SCSI block device.

**RETURNS** The status of the request, or **ERROR** if the request is unsupported.

**SEE ALSO** `scsiLib`

---

## scsiLoadUnit()

**NAME** `scsiLoadUnit()` – issue a LOAD/UNLOAD command to a SCSI device

**SYNOPSIS**

```
STATUS scsiLoadUnit
(
    SCSI_SEQ_DEV * pScsiSeqDev, /* ptr to SCSI physical device */
    BOOL          load,         /* TRUE=load, FALSE=unload */
    BOOL          reten,       /* TRUE=retention and unload */
    BOOL          eot          /* TRUE=end of tape and unload */
)
```

**DESCRIPTION** This routine issues a LOAD/UNLOAD command to a specified SCSI device.

**RETURNS** OK, or **ERROR** if the command fails.

**SEE ALSO** `scsiSeqLib`

---

## scsiMgrBusReset()

**NAME** scsiMgrBusReset() – handle a controller-bus reset event

**SYNOPSIS**

```
void scsiMgrBusReset
(
    SCSI_CTRL * pScsiCtrl    /* SCSI ctrlr on which bus reset */
)
```

**DESCRIPTION** This routine resets in turn: each attached physical device, each target, and the controller-finite-state machine. In practice, this routine implements the SCSI hard reset option.

---

**NOTE:** This routine does not physically reset the SCSI bus; see **scsiBusReset()**. This routine should not be called by application programs.

---

**RETURNS** N/A

**SEE ALSO** scsiMgrLib

---

## scsiMgrCtrlEvent()

**NAME** scsiMgrCtrlEvent() – send an event to the SCSI controller state machine

**SYNOPSIS**

```
void scsiMgrCtrlEvent
(
    SCSI_CTRL *    pScsiCtrl,
    SCSI_EVENT_TYPE eventType
)
```

**DESCRIPTION** This routine is called by the thread driver whenever selection, re-selection, or disconnection occurs or when a thread is activated. It manages a simple finite-state machine for the SCSI controller.

---

**NOTE:** This function should not be called by application programs.

---

**RETURNS** N/A

**SEE ALSO** scsiMgrLib

---

## scsiMgrEventNotify()

**NAME** `scsiMgrEventNotify()` – notify the SCSI manager of a SCSI (controller) event

**SYNOPSIS**

```
STATUS scsiMgrEventNotify
(
    SCSI_CTRL * pScsiCtrl,    /* pointer to SCSI controller structure */
    SCSI_EVENT * pEvent,     /* pointer to the SCSI event */
    int        eventSize    /* size of the event information */
)
```

**DESCRIPTION** This routine posts an event message on the appropriate SCSI manager queue, then notifies the SCSI manager that there is a message to be accepted.

---

**NOTE:** This routine should not be called by application programs.

---

No access serialization is required, because event messages are only posted by the SCSI controller ISR. See the reference entry for `scsiBusResetNotify()`.

**RETURNS** OK, or `ERROR` if the SCSI manager's event queue is full.

**SEE ALSO** `scsiMgrLib`, `scsiBusResetNotify()`

---

## scsiMgrShow()

**NAME** `scsiMgrShow()` – show status information for the SCSI manager

**SYNOPSIS**

```
void scsiMgrShow
(
    SCSI_CTRL * pScsiCtrl,    /* SCSI controller to use */
    BOOL      showPhysDevs,  /* TRUE => show phys dev details */
    BOOL      showThreads,   /* TRUE => show thread details */
    BOOL      showFreeThreads /* TRUE => show free thread IDs */
)
```

**DESCRIPTION** This routine shows the current state of the SCSI manager for the specified controller, including the total number of threads created and the number of threads currently free.

Optionally, this routine also shows details for all created physical devices on this controller and all threads for which SCSI requests are outstanding. It also shows the IDs of all free threads.

---

**NOTE:** The information displayed is volatile; this routine is best used when there is no activity on the SCSI bus. Threads allocated by a client but for which there are no outstanding SCSI requests are not shown.

---

**RETURNS** N/A

**SEE ALSO** `scsiMgrLib`

---

## scsiMgrThreadEvent()

**NAME** `scsiMgrThreadEvent()` – send an event to the thread state machine

**SYNOPSIS**

```
void scsiMgrThreadEvent
(
    SCSI_THREAD *      pThread,
    SCSI_THREAD_EVENT_TYPE eventType
)
```

**DESCRIPTION** This routine forwards an event to the thread's physical device. If the event is completion or deferral, it frees up the tag which was allocated when the thread was activated and either completes or defers the thread.

---

**NOTE:** This function should not be called by application programs.

---

The thread passed into this function does not have to be an active client thread (it may be an identification thread).

If the thread has no corresponding physical device, this routine does nothing. (This occasionally occurs if an unexpected disconnection or bus reset happens when an identification thread has not yet identified which physical device it corresponds to.

**RETURNS** N/A

**SEE ALSO** `scsiMgrLib`

---

## scsiModeSelect( )

**NAME** `scsiModeSelect()` – issue a MODE\_SELECT command to a SCSI device

**SYNOPSIS**

```
STATUS scsiModeSelect
(
    SCSI_PHYS_DEV * pScsiPhysDev, /* ptr to SCSI physical device */
    int           pageFormat,     /* value of the page format bit (0-1) */
    int           saveParams,     /* value of the save parameters bit (0-1) */
    char *       buffer,         /* ptr to output data buffer */
    int           bufLength      /* length of buffer in bytes */
)
```

**DESCRIPTION** This routine issues a MODE\_SELECT command to a specified SCSI device.

**RETURNS** OK, or ERROR if the command fails.

**SEE ALSO** `scsiLib`

---

## scsiModeSense( )

**NAME** `scsiModeSense()` – issue a MODE\_SENSE command to a SCSI device

**SYNOPSIS**

```
STATUS scsiModeSense
(
    SCSI_PHYS_DEV * pScsiPhysDev, /* ptr to SCSI physical device */
    int           pageControl,     /* value of the page control field (0-3) */
    int           pageCode,       /* value of the page code field (0-0x3f) */
    char *       buffer,         /* ptr to input data buffer */
    int           bufLength      /* length of buffer in bytes */
)
```

**DESCRIPTION** This routine issues a MODE\_SENSE command to a specified SCSI device.

**RETURNS** OK, or ERROR if the command fails.

**SEE ALSO** `scsiLib`



---

## scsiMsgInComplete()

**NAME** scsiMsgInComplete() – handle a complete SCSI message received from the target

**SYNOPSIS**

```
STATUS scsiMsgInComplete
(
    SCSI_CTRL * pScsiCtrl, /* ptr to SCSI controller info */
    SCSI_THREAD * pThread /* ptr to thread info */
)
```

**DESCRIPTION** This routine parses the complete message and takes any necessary action, which may include setting up an outgoing message in reply. If the message is not understood, the routine rejects it and returns an **ERROR** status.

---

**NOTE:** This function is intended for use only by SCSI controller drivers.

---

**RETURNS** OK, or **ERROR** if the message is not supported.

**SEE ALSO** scsi2Lib

---

## scsiMsgOutComplete()

**NAME** scsiMsgOutComplete() – perform post-processing after a SCSI message is sent

**SYNOPSIS**

```
STATUS scsiMsgOutComplete
(
    SCSI_CTRL * pScsiCtrl, /* ptr to SCSI controller info */
    SCSI_THREAD * pThread /* ptr to thread info */
)
```

**DESCRIPTION** This routine parses the complete message and takes any necessary action.

---

**NOTE:** This function is intended for use only by SCSI controller drivers.

---

**RETURNS** OK, or **ERROR** if the message is not supported.

**SEE ALSO** scsi2Lib

---

## scsiMsgOutReject()

|                    |                                                                                                                                                      |
|--------------------|------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>        | <code>scsiMsgOutReject()</code> – perform post-processing when an outgoing message is rejected                                                       |
| <b>SYNOPSIS</b>    | <pre>void scsiMsgOutReject (     SCSI_CTRL * pScsiCtrl, /* ptr to SCSI controller info */     SCSI_THREAD * pThread /* ptr to thread info */ )</pre> |
| <b>DESCRIPTION</b> | <b>NOTE:</b> This function is intended for use only by SCSI controller drivers.                                                                      |
| <b>RETURNS</b>     | OK, or ERROR if the message is not supported.                                                                                                        |
| <b>SEE ALSO</b>    | <code>scsi2Lib</code>                                                                                                                                |

---

## scsiPhysDevCreate()

|                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
|--------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>        | <code>scsiPhysDevCreate()</code> – create a SCSI physical device structure                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| <b>SYNOPSIS</b>    | <pre>SCSI_PHYS_DEV * scsiPhysDevCreate (     SCSI_CTRL * pScsiCtrl, /* ptr to SCSI controller info */     int devBusId, /* device's SCSI bus ID */     int devLUN, /* device's logical unit number */     int reqSenseLength, /* length of REQUEST SENSE data dev returns */     int devType, /* type of SCSI device */     BOOL removable, /* whether medium is removable */     int numBlocks, /* number of blocks on device */     int blockSize /* size of a block in bytes */ )</pre>                                              |
| <b>DESCRIPTION</b> | <p>This routine enables access to a SCSI device and must be the first routine invoked. It must be called once for each physical device on the SCSI bus.</p> <p>If <i>reqSenseLength</i> is NULL (0), one or more REQUEST_SENSE commands are issued to the device to determine the number of bytes of sense data it typically returns. Note that if the device returns variable amounts of sense data depending on its state, you must consult the device manual to determine the maximum amount of sense data that can be returned.</p> |

**scsiPhysDevDelete()**

If *devType* is NONE (-1), an INQUIRY command is issued to determine the device type; as an added benefit, it acquires the device's make and model number. The **scsiShow()** routine displays this information. Common values of *devType* can be found in **scsiLib.h** or in the SCSI specification.

If *numBlocks* or *blockSize* are specified as NULL (0), a READ\_CAPACITY command is issued to determine those values. This occurs only for device types that support READ\_CAPACITY.

**RETURNS** A pointer to the created SCSI\_PHYS\_DEV structure, or NULL if the routine is unable to create the physical-device structure.

**SEE ALSO** **scsiLib**

**scsiPhysDevDelete()**

**NAME** **scsiPhysDevDelete()** – delete a SCSI physical-device structure

**SYNOPSIS**

```
STATUS scsiPhysDevDelete
(
    SCSI_PHYS_DEV * pScsiPhysDev /* ptr to SCSI physical device info */
)
```

**DESCRIPTION** This routine deletes a specified SCSI physical-device structure.

**RETURNS** OK, or ERROR if **pScsiPhysDev** is NULL or SCSI\_BLK\_DEVS have been created on the device.

**SEE ALSO** **scsiLib**

**scsiPhysDevIdGet()**

**NAME** **scsiPhysDevIdGet()** – return a pointer to a SCSI\_PHYS\_DEV structure

**SYNOPSIS**

```
SCSI_PHYS_DEV * scsiPhysDevIdGet
(
    SCSI_CTRL * pScsiCtrl, /* ptr to SCSI controller info */
    int        devBusId, /* device's SCSI bus ID */
)
```

```
int devLUN /* device's logical unit number */
)
```

- DESCRIPTION** This routine returns a pointer to the `SCSI_PHYS_DEV` structure of the SCSI physical device located at a specified bus ID (*devBusId*) and logical unit number (*devLUN*) and attached to a specified SCSI controller (*pScsiCtrl*).
- RETURNS** A pointer to the specified `SCSI_PHYS_DEV` structure, or `NULL` if the structure does not exist.
- SEE ALSO** `scsiLib`

## scsiPhysDevShow()

**NAME** `scsiPhysDevShow()` – show status information for a physical device

**SYNOPSIS**

```
void scsiPhysDevShow
(
    SCSI_PHYS_DEV * pScsiPhysDev, /* physical device to be displayed */
    BOOL          showThreads, /* show IDs of associated threads */
    BOOL          noHeader     /* do not print title line */
)
```

**DESCRIPTION** This routine shows the state, the current nexus type, the current tag number, the number of tagged commands in progress, and the number of waiting and active threads for a SCSI physical device. Optionally, it shows the IDs of waiting and active threads, if any. This routine may be called at any time, but note that all of the information displayed is volatile.

**RETURNS** N/A

**SEE ALSO** `scsi2Lib`



---

## scsiRdSecs()

**NAME** scsiRdSecs() – read sector(s) from a SCSI block device

**SYNOPSIS**

```
STATUS scsiRdSecs
(
    SCSI_BLK_DEV * pScsiBlkDev, /* ptr to SCSI block device info */
    int          sector,        /* sector number to be read */
    int          numSecs,       /* total sectors to be read */
    char *       buffer         /* ptr to input data buffer */
)
```

**DESCRIPTION** This routine reads the specified physical sector(s) from a specified physical device.

**RETURNS** OK, or ERROR if the sector(s) cannot be read.

**SEE ALSO** scsiLib

---

## scsiRdTape()

**NAME** scsiRdTape() – read bytes or blocks from a SCSI tape device

**SYNOPSIS**

```
int scsiRdTape
(
    SCSI_SEQ_DEV * pScsiSeqDev, /* ptr to SCSI sequential device info */
    UINT          count,        /* total bytes or blocks to be read */
    char *        buffer,       /* ptr to input data buffer */
    BOOL          fixedSize     /* if variable size blocks */
)
```

**DESCRIPTION** This routine reads the specified number of bytes or blocks from a specified physical device. If the boolean *fixedSize* is true, then *numBytes* represents the number of blocks of size *blockSize*, defined in the **pScsiPhysDev** structure. If variable block sizes are used (*fixedSize* = FALSE), then *numBytes* represents the actual number of bytes to be read.

**RETURNS** Number of bytes or blocks actually read, 0 if EOF, or ERROR.

**SEE ALSO** scsiSeqLib

---

## scsiReadCapacity()

**NAME** `scsiReadCapacity()` – issue a `READ_CAPACITY` command to a SCSI device

**SYNOPSIS**

```
STATUS scsiReadCapacity  
(  
    SCSI_PHYS_DEV * pScsiPhysDev, /* ptr to SCSI physical device */  
    int *          pLastLBA,      /* where to return last logical block */  
                                /* address */  
    int *          pBlkLength     /* where to return block length */  
)
```

**DESCRIPTION** This routine issues a `READ_CAPACITY` command to a specified SCSI device.

**RETURNS** `OK`, or `ERROR` if the command fails.

**SEE ALSO** `scsiLib`

---

## scsiRelease()

**NAME** `scsiRelease()` – issue a `RELEASE` command to a SCSI device

**SYNOPSIS**

```
STATUS scsiRelease  
(  
    SCSI_PHYS_DEV * pScsiPhysDev /* ptr to SCSI physical device */  
)
```

**DESCRIPTION** This routine issues a `RELEASE` command to a specified SCSI device.

**RETURNS** `OK`, or `ERROR` if the command fails.

**SEE ALSO** `scsiDirectLib`

## scsiReleaseUnit()

**NAME** scsiReleaseUnit() – issue a RELEASE UNIT command to a SCSI device

**SYNOPSIS**

```
STATUS scsiReleaseUnit
(
    SCSI_SEQ_DEV * pScsiSeqDev /* ptr to SCSI sequential device */
)
```

**DESCRIPTION** This routine issues a RELEASE UNIT command to a specified SCSI device.

**RETURNS** OK, or ERROR if the command fails.

**SEE ALSO** scsiSeqLib

---

## scsiReqSense()

**NAME** scsiReqSense() – issue a REQUEST\_SENSE command to a SCSI device and read results

**SYNOPSIS**

```
STATUS scsiReqSense
(
    SCSI_PHYS_DEV * pScsiPhysDev, /* ptr to SCSI physical device */
    char *          buffer,        /* ptr to input data buffer */
    int             bufLength      /* length of buffer in bytes */
)
```

**DESCRIPTION** This routine issues a REQUEST\_SENSE command to a specified SCSI device and reads the results.

**RETURNS** OK, or ERROR if the command fails.

**SEE ALSO** scsiLib

---

## scsiReserve()

**NAME** `scsiReserve()` – issue a RESERVE command to a SCSI device

**SYNOPSIS**

```
STATUS scsiReserve  
(  
    SCSI_PHYS_DEV * pScsiPhysDev /* ptr to SCSI physical device */  
)
```

**DESCRIPTION** This routine issues a RESERVE command to a specified SCSI device.

**RETURNS** OK, or ERROR if the command fails.

**SEE ALSO** `scsiDirectLib`

---

## scsiReserveUnit()

**NAME** `scsiReserveUnit()` – issue a RESERVE UNIT command to a SCSI device

**SYNOPSIS**

```
STATUS scsiReserveUnit  
(  
    SCSI_SEQ_DEV * pScsiSeqDev /* ptr to SCSI sequential device */  
)
```

**DESCRIPTION** This routine issues a RESERVE UNIT command to a specified SCSI device.

**RETURNS** OK, or ERROR if the command fails.

**SEE ALSO** `scsiSeqLib`

---

## scsiRewind()

**NAME** scsiRewind() – issue a REWIND command to a SCSI device

**SYNOPSIS**

```
STATUS scsiRewind  
(  
    SCSI_SEQ_DEV * pScsiSeqDev /* ptr to SCSI Sequential device */  
)
```

**DESCRIPTION** This routine issues a REWIND command to a specified SCSI device.

**RETURNS** OK, or ERROR if the command fails.

**SEE ALSO** scsiSeqLib

---

## scsiSeqDevCreate()

**NAME** scsiSeqDevCreate() – create a SCSI sequential device

**SYNOPSIS**

```
SEQ_DEV *scsiSeqDevCreate  
(  
    SCSI_PHYS_DEV * pScsiPhysDev /* ptr to SCSI physical device info */  
)
```

**DESCRIPTION** This routine creates a SCSI sequential device and saves a pointer to this SEQ\_DEV in the SCSI physical device. The following functions are initialized in this structure:

|                    |                          |
|--------------------|--------------------------|
| sd_seqRd           | scsiRdTape()             |
| sd_seqWrt          | scsiWrtTape()            |
| sd_ioctl           | scsiIoctl() (in scsiLib) |
| sd_seqWrtFileMarks | scsiWrtFileMarks()       |
| sd_statusChk       | scsiSeqStatusCheck()     |
| sd_reset           | (not used)               |
| sd_rewind          | scsiRewind()             |
| sd_reserve         | scsiReserve()            |
| sd_release         | scsiRelease()            |
| sd_readBlkLim      | scsiSeqReadBlockLimits() |
| sd_load            | scsiLoadUnit()           |
| sd_space           | scsiSpace()              |
| sd_erase           | scsiErase()              |

Only one `SEQ_DEV` per `SCSI_PHYS_DEV` is allowed, unlike `BLK_DEVS` where an entire list is maintained. Therefore, this routine can be called only once per creation of a sequential device.

**RETURNS** A pointer to the `SEQ_DEV` structure, or `NULL` if the command fails.

**SEE ALSO** `scsiSeqLib`

## `scsiSeqIoctl()`

**NAME** `scsiSeqIoctl()` – perform an I/O control function for sequential access devices

**SYNOPSIS**

```
int scsiSeqIoctl
(
    SCSI_SEQ_DEV * pScsiSeqDev, /* ptr to SCSI sequential device */
    int          function,      /* ioctl function code */
    int          arg            /* argument to pass to called function */
)
```

**DESCRIPTION** This routine issues `scsiSeqLib` commands to perform sequential device-specific I/O control operations.

**RETURNS** `OK` or `ERROR`.

**ERRNO** `S_scsiLib_INVALID_BLOCK_SIZE`

**SEE ALSO** `scsiSeqLib`

## `scsiSeqReadBlockLimits()`

**NAME** `scsiSeqReadBlockLimits()` – issue a `READ_BLOCK_LIMITS` command to a SCSI device

**SYNOPSIS**

```
STATUS scsiSeqReadBlockLimits
(
    SCSI_SEQ_DEV * pScsiSeqDev, /* ptr to SCSI sequential device */
    int *         pMaxBlockLength, /* where to return maximum block length */
    UINT16 *     pMinBlockLength /* where to return minimum block length */
)
```

**DESCRIPTION** This routine issues a **READ\_BLOCK\_LIMITS** command to a specified SCSI device.

**RETURNS** **OK**, or **ERROR** if the command fails.

**SEE ALSO** **scsiSeqLib**

---

## scsiSeqStatusCheck( )

**NAME** **scsiSeqStatusCheck( )** – detect a change in media

**SYNOPSIS**

```
STATUS scsiSeqStatusCheck  
(  
    SCSI_SEQ_DEV * pScsiSeqDev /* ptr to a sequential dev */  
)
```

**DESCRIPTION** This routine issues a **TEST\_UNIT\_READY** command to a SCSI device to detect a change in media. It is called by file systems before executing **open( )** or **creat( )**.

**RETURNS** **OK** or **ERROR**.

**SEE ALSO** **scsiSeqLib**

---

## scsiShow( )

**NAME** **scsiShow( )** – list the physical devices attached to a SCSI controller

**SYNOPSIS**

```
STATUS scsiShow  
(  
    SCSI_CTRL * pScsiCtrl /* ptr to SCSI controller info */  
)
```

**DESCRIPTION** This routine displays the SCSI bus ID, logical unit number (LUN), vendor ID, product ID, firmware revision (rev.), device type, number of blocks, block size in bytes, and a pointer to the associated **SCSI\_PHYS\_DEV** structure for each physical SCSI device known to be attached to a specified SCSI controller.

---

**NOTE:** If *pScsiCtrl* is **NULL**, the value of the global variable **pSysScsiCtrl** is used, unless it is also **NULL**.

---

**RETURNS** OK, or **ERROR** if both *pScsiCtrl* and **pSysScsiCtrl** are **NULL**.

**SEE ALSO** **scsiLib**

---

## scsiSpace()

**NAME** **scsiSpace()** – move the tape on a specified physical SCSI device

**SYNOPSIS**

```

STATUS scsiSpace
(
    SCSI_SEQ_DEV * pScsiSeqDev, /* ptr to SCSI sequential device info */
    int          count,         /* count for space command */
    int          spaceCode     /* code for the type of space command */
)

```

**DESCRIPTION** This routine moves the tape on a specified SCSI physical device. There are two types of space code that are mandatory in SCSI; currently these are the only two supported:

| <b>Code</b> | <b>Description</b>    | <b>Support</b> |
|-------------|-----------------------|----------------|
| 000         | Blocks                | Yes            |
| 001         | File marks            | Yes            |
| 010         | Sequential file marks | No             |
| 011         | End-of-data           | No             |
| 100         | Set marks             | No             |
| 101         | Sequential set marks  | No             |

**RETURNS** OK, or **ERROR** if an error is returned by the device.

**ERRNO** **S\_scsiLib\_ILLEGAL\_REQUEST**

**SEE ALSO** **scsiSeqLib**

---

## scsiStartStopUnit()

|                    |                                                                                                                                                                                  |
|--------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>        | scsiStartStopUnit() – issue a START_STOP_UNIT command to a SCSI device                                                                                                           |
| <b>SYNOPSIS</b>    | <pre>STATUS scsiStartStopUnit (     SCSI_PHYS_DEV * pScsiPhysDev, /* ptr to SCSI physical device */     BOOL          start           /* TRUE == start, FALSE == stop */ )</pre> |
| <b>DESCRIPTION</b> | This routine issues a START_STOP_UNIT command to a specified SCSI device.                                                                                                        |
| <b>RETURNS</b>     | OK, or ERROR if the command fails.                                                                                                                                               |
| <b>SEE ALSO</b>    | scsiDirectLib                                                                                                                                                                    |

---

## scsiSyncXferNegotiate()

|                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
|--------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>        | scsiSyncXferNegotiate() – initiate or continue negotiating transfer parameters                                                                                                                                                                                                                                                                                                                                                                                                                   |
| <b>SYNOPSIS</b>    | <pre>void scsiSyncXferNegotiate (     SCSI_CTRL *      pScsiCtrl, /* ptr to SCSI controller info */     SCSI_TARGET *   pScsiTarget, /* ptr to SCSI target info */     SCSI_SYNC_XFER_EVENT eventType /* tells what has just happened */ )</pre>                                                                                                                                                                                                                                                 |
| <b>DESCRIPTION</b> | <p>This routine manages negotiation by means of a finite-state machine which is driven by “significant events” such as incoming and outgoing messages. Each SCSI target has its own independent state machine.</p> <hr/> <p><b>NOTE:</b> If the controller does not support synchronous transfer or if the target’s maximum REQ/ACK offset is zero, attempts to initiate a round of negotiation are ignored.</p> <hr/> <p>This function is intended for use only by SCSI controller drivers.</p> |
| <b>RETURNS</b>     | N/A                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| <b>SEE ALSO</b>    | scsi2Lib                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |

---

## scsiTapeModeSelect( )

**NAME** `scsiTapeModeSelect()` – issue a `MODE_SELECT` command to a SCSI tape device

**SYNOPSIS**

```
STATUS scsiTapeModeSelect
(
    SCSI_PHYS_DEV * pScsiPhysDev, /* ptr to SCSI physical device */
    int           pageFormat,     /* value of the page format bit (0-1) */
    int           saveParams,     /* value of the save parameters bit (0-1) */
    char *       buffer,         /* ptr to output data buffer */
    int           bufLength      /* length of buffer in bytes */
)
```

**DESCRIPTION** This routine issues a `MODE_SELECT` command to a specified SCSI device.

**RETURNS** OK, or ERROR if the command fails.

**SEE ALSO** `scsiSeqLib`

---

## scsiTapeModeSense( )

**NAME** `scsiTapeModeSense()` – issue a `MODE_SENSE` command to a SCSI tape device

**SYNOPSIS**

```
STATUS scsiTapeModeSense
(
    SCSI_PHYS_DEV * pScsiPhysDev, /* ptr to SCSI physical device */
    int           pageControl,     /* value of the page control field (0-3) */
    int           pageCode,       /* value of the page code field (0-0x3f) */
    char *       buffer,         /* ptr to input data buffer */
    int           bufLength      /* length of buffer in bytes */
)
```

**DESCRIPTION** This routine issues a `MODE_SENSE` command to a specified SCSI tape device.

**RETURNS** OK, or ERROR if the command fails.

**SEE ALSO** `scsiSeqLib`

S

---

## scsiTargetOptionsGet()

**NAME** scsiTargetOptionsGet() – get options for one or all SCSI targets

**SYNOPSIS**

```
STATUS scsiTargetOptionsGet
(
    SCSI_CTRL *   pScsiCtrl, /* ptr to SCSI controller info */
    int          devBusId, /* target to interrogate */
    SCSI_OPTIONS * pOptions /* buffer to return options */
)
```

**DESCRIPTION** This routine copies the current options for the specified target into the caller's buffer.

**RETURNS** OK, or ERROR if the bus ID is invalid.

**SEE ALSO** scsi2Lib

---

## scsiTargetOptionsSet()

**NAME** scsiTargetOptionsSet() – set options for one or all SCSI targets

**SYNOPSIS**

```
STATUS scsiTargetOptionsSet
(
    SCSI_CTRL *   pScsiCtrl, /* ptr to SCSI controller info */
    int          devBusId, /* target to affect, or all */
    SCSI_OPTIONS * pOptions, /* buffer containing new options */
    UINT         which      /* which options to change */
)
```

**DESCRIPTION** This routine sets the options defined by the bit mask **which** for the specified target (or all targets if **devBusId** is SCSI\_SET\_OPT\_ALL\_TARGETS).

The bit mask **which** can be any combination of the following, bitwise OR'd together (corresponding fields in the SCSI\_OPTIONS structure are shown in parentheses):

|                          |            |                                     |
|--------------------------|------------|-------------------------------------|
| SCSI_SET_OPT_TIMEOUT     | selTimeOut | select timeout period, microseconds |
| SCSI_SET_OPT_MESSAGES    | messages   | FALSE to disable SCSI messages      |
| SCSI_SET_OPT_DISCONNECT  | disconnect | FALSE to disable discon/recon       |
| SCSI_SET_OPT_XFER_PARAMS | maxOffset, | max sync xfer offset, 0>async       |
|                          | minPeriod  | min sync xfer period, x 4 nsec.     |

|                          |                  |                               |
|--------------------------|------------------|-------------------------------|
| SCSI_SET_OPT_TAG_PARAMS  | <b>tagType,</b>  | default tag type (SCSI_TAG_*) |
|                          | <b>maxTags</b>   | max cmd tags available        |
| SCSI_SET_OPT_WIDE_PARAMS | <b>xferWidth</b> | data transfer width setting.  |
|                          |                  | xferWidth = 0 ; 8 bits wide   |
|                          |                  | xferWidth = 1 ; 16 bits wide  |

---

**NOTE:** This routine can be used after the target device has already been used; in this case, however, it is not possible to change the tag parameters. This routine must not be used while there is any SCSI activity on the specified target(s).

---

**RETURNS** OK, or **ERROR** if the bus ID or options are invalid.

**SEE ALSO** scsi2Lib

---

## scsiTargetOptionsShow( )

**NAME** scsiTargetOptionsShow( ) – display options for specified SCSI target

**SYNOPSIS**

```
STATUS scsiTargetOptionsShow
(
    SCSI_CTRL * pScsiCtrl,    /* ptr to SCSI controller info */
    int         devBusId      /* target to interrogate */
)
```

**DESCRIPTION** This routine displays the current target options for the specified target in the following format:

Target Options (id *scsi bus ID*):  
selection Timeout: *timeout* nano secs  
messages allowed: **TRUE** or **FALSE**  
disconnect allowed: **TRUE** or **FALSE**  
REQ/ACK offset: *negotiated offset*  
transfer period: *negotiated period*  
transfer width: 8 or 16 bits maximum transfer rate: *peak transfer rate* MB/sec  
tag type: *tag type*  
maximum tags: *max tags*

**RETURNS** OK, or **ERROR** if the bus ID is invalid.

**SEE ALSO** scsi2Lib



## scsiTestUnitRdy()

|                    |                                                                                                          |
|--------------------|----------------------------------------------------------------------------------------------------------|
| <b>NAME</b>        | <code>scsiTestUnitRdy()</code> – issue a TEST_UNIT_READY command to a SCSI device                        |
| <b>SYNOPSIS</b>    | <pre>STATUS scsiTestUnitRdy (     SCSI_PHYS_DEV * pScsiPhysDev /* ptr to SCSI physical device */ )</pre> |
| <b>DESCRIPTION</b> | This routine issues a TEST_UNIT_READY command to a specified SCSI device.                                |
| <b>RETURNS</b>     | OK, or ERROR if the command fails.                                                                       |
| <b>SEE ALSO</b>    | <code>scsiLib</code>                                                                                     |

---

## scsiThreadInit()

|                    |                                                                                                                                                                                                                                                                |
|--------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>        | <code>scsiThreadInit()</code> – perform generic SCSI thread initialization                                                                                                                                                                                     |
| <b>SYNOPSIS</b>    | <pre>STATUS scsiThreadInit (     SCSI_THREAD * pThread )</pre>                                                                                                                                                                                                 |
| <b>DESCRIPTION</b> | This routine initializes the controller-independent parts of a thread structure, which are specific to the SCSI manager.<br><br><b>NOTE:</b> This function should not be called by application programs. It is intended to be used by SCSI controller drivers. |
| <b>RETURNS</b>     | OK, or ERROR if the thread cannot be initialized.                                                                                                                                                                                                              |
| <b>SEE ALSO</b>    | <code>scsi2Lib</code>                                                                                                                                                                                                                                          |

---

## scsiWideXferNegotiate()

**NAME** `scsiWideXferNegotiate()` – initiate or continue negotiating wide parameters

**SYNOPSIS**

```
void scsiWideXferNegotiate
(
    SCSI_CTRL *          pScsiCtrl, /* ptr to SCSI controller info */
    SCSI_TARGET *       pScsiTarget, /* ptr to SCSI target info */
    SCSI_WIDE_XFER_EVENT eventType /* tells what has just happened */
)
```

**DESCRIPTION** This routine manages negotiation means of a finite-state machine which is driven by “significant events” such as incoming and outgoing messages. Each SCSI target has its own independent state machine.

---

**NOTE:** If the controller does not support wide transfers or the target’s transfer width is zero, attempts to initiate a round of negotiation are ignored; this is because zero is the default narrow transfer.

---

This function is intended for use only by SCSI controller drivers.

**RETURNS** N/A

**SEE ALSO** `scsi2Lib`

---

## scsiWrtFileMarks()

**NAME** `scsiWrtFileMarks()` – write file marks to a SCSI sequential device

**SYNOPSIS**

```
STATUS scsiWrtFileMarks
(
    SCSI_SEQ_DEV * pScsiSeqDev, /* ptr to SCSI sequential device info */
    int            numMarks,    /* number of file marks to write */
    BOOL           shortMark    /* TRUE to write short file mark */
)
```

**DESCRIPTION** This routine writes file marks to a specified physical device.

**RETURNS** OK, or ERROR if the file mark cannot be written.

**SEE ALSO** `scsiSeqLib`

---

## scsiWrtSecs()

**NAME** scsiWrtSecs() – write sector(s) to a SCSI block device

**SYNOPSIS**

```
STATUS scsiWrtSecs
(
    SCSI_BLK_DEV * pScsiBlkDev, /* ptr to SCSI block device info */
    int          sector,        /* sector number to be written */
    int          numSecs,       /* total sectors to be written */
    char *       buffer         /* ptr to input data buffer */
)
```

**DESCRIPTION** This routine writes the specified physical sector(s) to a specified physical device.

**RETURNS** OK, or ERROR if the sector(s) cannot be written.

**SEE ALSO** scsiLib

---

## scsiWrtTape()

**NAME** scsiWrtTape() – write data to a SCSI tape device

**SYNOPSIS**

```
STATUS scsiWrtTape
(
    SCSI_SEQ_DEV * pScsiSeqDev, /* ptr to SCSI sequential device info */
    int          numBytes,      /* total bytes or blocks to be written */
    char *       buffer,        /* ptr to input data buffer */
    BOOL         fixedSize      /* if variable size blocks */
)
```

**DESCRIPTION** This routine writes data to the current block on a specified physical device. If the boolean *fixedSize* is true, then *numBytes* represents the number of blocks of size *blockSize*, defined in the **pScsiPhysDev** structure. If variable block sizes are used (*fixedSize* = FALSE), then *numBytes* represents the actual number of bytes to be written. If *numBytes* is greater than the **maxBytesLimit** field defined in the **pScsiPhysDev** structure, then more than one SCSI transaction is used to transfer the data.

**RETURNS** OK, or ERROR if the data cannot be written or zero bytes are written.

**SEE ALSO** scsiSeqLib

---

## select()

**NAME** select() – pend on a set of file descriptors

**SYNOPSIS**

```
int select
(
    int          width,          /* number of bits to examine from 0 */
    fd_set *    pReadFds,       /* read fds */
    fd_set *    pWriteFds,      /* write fds */
    fd_set *    pExceptFds,     /* exception fds (unsupported) */
    struct timeval * pTimeout    /* max time to wait, NULL = forever */
)
```

**DESCRIPTION** This routine permits a task to pend until one of a set of file descriptors becomes ready. Three parameters -- *pReadFds*, *pWriteFds*, and *pExceptFds* -- point to file descriptor sets in which each bit corresponds to a particular file descriptor. Bits set in the read file descriptor set (*pReadFds*) will cause **select()** to pend until data is available on any of the corresponding file descriptors, while bits set in the write file descriptor set (*pWriteFds*) will cause **select()** to pend until any of the corresponding file descriptors become writable. (The *pExceptFds* parameter is currently unused, but is provided for UNIX call compatibility.)

The following macros are available for setting the appropriate bits in the file descriptor set structure:

```
FD_SET(fd, &fdset)
FD_CLR(fd, &fdset)
FD_ZERO(&fdset)
```

If either *pReadFds* or *pWriteFds* is **NULL**, they are ignored. The *width* parameter defines how many bits will be examined in the file descriptor sets, and should be set to either the maximum file descriptor value in use plus one, or simply to **FD\_SETSIZE**. When **select()** returns, it zeros out the file descriptor sets, and sets only the bits that correspond to file descriptors that are ready. The **FD\_ISSET** macro may be used to determine which bits are set.

If *pTimeout* is **NULL**, **select()** will block indefinitely. If *pTimeout* is not **NULL**, but points to a **timeval** structure with an effective time of zero, the file descriptors in the file descriptor sets will be polled and the results returned immediately. If the effective time value is greater than zero, **select()** will return after the specified time has elapsed, even if none of the file descriptors are ready.

Applications can use **select()** with pipes and serial devices, in addition to sockets. Also, **select()** now examines write file descriptors in addition to read file descriptors; however, exception file descriptors remain unsupported.

## ***selectInit()***

The value for the maximum number of file descriptors configured in the system (NUM\_FILES) should be less than or equal to the value of **FD\_SETSIZE** (2048).

Driver developers should consult the *VxWorks Programmer's Guide: I/O System* for details on writing drivers that will use **select()**.

**RETURNS** The number of file descriptors with activity, 0 if timed out, or **ERROR** if an error occurred when the driver's **select()** routine was invoked via **ioctl()**.

**ERRNOS** Possible **errno**s generated by this routine include:

**S\_selectLib\_NO\_SELECT\_SUPPORT\_IN\_DRIVER**

A driver associated with one or more *fds* does not support **select()**.

**S\_selectLib\_NO\_SELECT\_CONTEXT**

The task's select context was not initialized at task creation time.

**S\_selectLib\_WIDTH\_OUT\_OF\_RANGE**

The width parameter is greater than the maximum possible *fd*.

**SEE ALSO** **selectLib**, *VxWorks Programmer's Guide: I/O System*

---

## **selectInit()**

**NAME** **selectInit()** – initialize the select facility

**SYNOPSIS**

```
void selectInit
(
    int numFiles          /* maximum number of open files */
)
```

**DESCRIPTION** This routine initializes the UNIX BSD 4.3 select facility. It should be called only once, and typically is called from the root task, **usrRoot()**, in **usrConfig.c**. It installs a task create hook such that a select context is initialized for each task.

**RETURNS** N/A

**SEE ALSO** **selectLib**

---

## selNodeAdd()

**NAME** `selNodeAdd()` – add a wake-up node to a `select()` wake-up list

**SYNOPSIS**

```
STATUS selNodeAdd
(
    SEL_WAKEUP_LIST * pWakeupList, /* list of tasks to wake up */
    SEL_WAKEUP_NODE * pWakeupNode /* node to add to list */
)
```

**DESCRIPTION** This routine adds a wake-up node to a device’s wake-up list. It is typically called from a driver’s `FIOSELECT` function.

**RETURNS** `OK`, or `ERROR` if memory is insufficient.

**SEE ALSO** `selectLib`

---

## selNodeDelete()

**NAME** `selNodeDelete()` – find and delete a node from a `select()` wake-up list

**SYNOPSIS**

```
STATUS selNodeDelete
(
    SEL_WAKEUP_LIST * pWakeupList, /* list of tasks to wake up */
    SEL_WAKEUP_NODE * pWakeupNode /* node to delete from list */
)
```

**DESCRIPTION** This routine deletes a specified wake-up node from a specified wake-up list. Typically, it is called by a driver’s `FIOUNSELECT` function.

**RETURNS** `OK`, or `ERROR` if the node is not found in the wake-up list.

**SEE ALSO** `selectLib`

S

---

## selWakeup()

**NAME** selWakeup() – wake up a task pending in **select()**

**SYNOPSIS**

```
void selWakeup
(
    SEL_WAKEUP_NODE * pWakeupNode /* node to wake up */
)
```

**DESCRIPTION** This routine wakes up a task pending in **select()**. Once a driver's **FIOSELECT** function installs a wake-up node in a device's wake-up list (using **selNodeAdd()**) and checks to make sure the device is ready, this routine ensures that the **select()** call does not pend.

**RETURNS** N/A

**SEE ALSO** **selectLib**

---

## selWakeupAll()

**NAME** selWakeupAll() – wake up all tasks in a **select()** wake-up list

**SYNOPSIS**

```
void selWakeupAll
(
    SEL_WAKEUP_LIST * pWakeupList, /* list of tasks to wake up */
    SELECT_TYPE      type          /* readers (SELREAD) or writers (SELWRITE) */
)
```

**DESCRIPTION** This routine wakes up all tasks pending in **select()** that are waiting for a device; it is called by a driver when the device becomes ready. The *type* parameter specifies the task to be awakened, either reader tasks (**SELREAD**) or writer tasks (**SELWRITE**).

**RETURNS** N/A

**SEE ALSO** **selectLib**

---

## selWakeupListInit()

|                    |                                                                                                                      |
|--------------------|----------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>        | <code>selWakeupListInit()</code> – initialize a <code>select()</code> wake-up list                                   |
| <b>SYNOPSIS</b>    | <pre>void selWakeupListInit (     SEL_WAKEUP_LIST * pWakeupList /* wake-up list to initialize */ )</pre>             |
| <b>DESCRIPTION</b> | This routine should be called in a device's create routine to initialize the <code>SEL_WAKEUP_LIST</code> structure. |
| <b>RETURNS</b>     | N/A                                                                                                                  |
| <b>SEE ALSO</b>    | <code>selectLib</code>                                                                                               |

---

## selWakeupListLen()

|                    |                                                                                                                                                                                                                                                                                            |
|--------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>        | <code>selWakeupListLen()</code> – get the number of nodes in a <code>select()</code> wake-up list                                                                                                                                                                                          |
| <b>SYNOPSIS</b>    | <pre>int selWakeupListLen (     SEL_WAKEUP_LIST * pWakeupList /* list of tasks to wake up */ )</pre>                                                                                                                                                                                       |
| <b>DESCRIPTION</b> | This routine returns the number of nodes in a specified <code>SEL_WAKEUP_LIST</code> . It can be used by a driver to determine if any tasks are currently pending in <code>select()</code> on this device, and whether these tasks need to be activated with <code>selWakeupAll()</code> . |
| <b>RETURNS</b>     | The number of nodes currently in a <code>select()</code> wake-up list, or <code>ERROR</code> .                                                                                                                                                                                             |
| <b>SEE ALSO</b>    | <code>selectLib</code>                                                                                                                                                                                                                                                                     |

## selWakeupListTerm()

|                    |                                                                                                           |
|--------------------|-----------------------------------------------------------------------------------------------------------|
| <b>NAME</b>        | selWakeupListTerm() – terminate a select() wake-up list                                                   |
| <b>SYNOPSIS</b>    | <pre>void selWakeupListTerm (     SEL_WAKEUP_LIST * pWakeupList /* wake-up list to terminate */ )</pre>   |
| <b>DESCRIPTION</b> | This routine should be called in a device's terminate routine to terminate the SEL_WAKEUP_LIST structure. |
| <b>RETURNS</b>     | N/A                                                                                                       |
| <b>SEE ALSO</b>    | selectLib                                                                                                 |

---

## selWakeupType()

|                    |                                                                                                                                                                                                |
|--------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>        | selWakeupType() – get the type of a select() wake-up node                                                                                                                                      |
| <b>SYNOPSIS</b>    | <pre>SELECT_TYPE selWakeupType (     SEL_WAKEUP_NODE * pWakeupNode /* node to get type of */ )</pre>                                                                                           |
| <b>DESCRIPTION</b> | This routine returns the type of a specified SEL_WAKEUP_NODE. It is typically used in a device's FIOSELECT function to determine if the device is being selected for read or write operations. |
| <b>RETURNS</b>     | SELREAD (read operation) or SELWRITE (write operation).                                                                                                                                        |
| <b>SEE ALSO</b>    | selectLib                                                                                                                                                                                      |

---

## semBCreate()

|                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
|--------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>        | <b>semBCreate()</b> – create and initialize a binary semaphore                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| <b>SYNOPSIS</b>    | <pre>SEM_ID semBCreate (     int          options,      /* semaphore options */     SEM_B_STATE initialState /* initial semaphore state */ )</pre>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
| <b>DESCRIPTION</b> | <p>This routine allocates and initializes a binary semaphore. The semaphore is initialized to the <i>initialState</i> of either <b>SEM_FULL</b> (1) or <b>SEM_EMPTY</b> (0).</p> <p>The <i>options</i> parameter specifies the queuing style for blocked tasks. Tasks can be queued on a priority basis or a first-in-first-out basis. These options are <b>SEM_Q_PRIORITY</b> (0x1) and <b>SEM_Q_FIFO</b> (0x0), respectively. That parameter also specifies if <b>semGive()</b> should return <b>ERROR</b> when the semaphore fails to send events. This option is turned off by default; it is activated by doing a bitwise-OR of <b>SEM_EVENTSEND_ERR_NOTIFY</b> (0x10) with the queuing style of the semaphore.</p> |
| <b>RETURNS</b>     | The semaphore ID, or <b>NULL</b> if memory cannot be allocated.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| <b>SEE ALSO</b>    | <b>semBLib</b>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |

---

## semBSmCreate()

|                    |                                                                                                                                                                                                                                                                                                                     |
|--------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>        | <b>semBSmCreate()</b> – create and initialize a shared memory binary semaphore (VxMP Opt.)                                                                                                                                                                                                                          |
| <b>SYNOPSIS</b>    | <pre>SEM_ID semBSmCreate (     int          options,      /* semaphore options */     SEM_B_STATE initialState /* initial semaphore state */ )</pre>                                                                                                                                                                |
| <b>DESCRIPTION</b> | <p>This routine allocates and initializes a shared memory binary semaphore. The semaphore is initialized to an <i>initialState</i> of either <b>SEM_FULL</b> (available) or <b>SEM_EMPTY</b> (not available). The shared semaphore structure is allocated from the shared semaphore dedicated memory partition.</p> |

The semaphore ID returned by this routine can be used directly by the generic semaphore-handling routines in **semLib** -- **semGive()**, **semTake()**, and **semFlush()** -- and the show routines, such as **show()** and **semShow()**.

The queuing style for blocked tasks is set by *options*; the only supported queuing style for shared memory semaphores is first-in-first-out, selected by **SEM\_Q\_FIFO**.

Before this routine can be called, the shared memory objects facility must be initialized (see **semSmLib**).

The maximum number of shared memory semaphores (binary plus counting) that can be created is **SM\_OBJ\_MAX\_SEM**, a configurable parameter.

|                     |                                                                                                                                            |
|---------------------|--------------------------------------------------------------------------------------------------------------------------------------------|
| <b>AVAILABILITY</b> | This routine is distributed as a component of the unbundled shared memory support option, VxMP.                                            |
| <b>RETURNS</b>      | The semaphore ID, or <b>NULL</b> if memory cannot be allocated from the shared semaphore dedicated memory partition.                       |
| <b>ERRNO</b>        | <b>S_memLib_NOT_ENOUGH_MEMORY</b> , <b>S_semLib_INVALID_QUEUE_TYPE</b> ,<br><b>S_semLib_INVALID_STATE</b> , <b>S_smObjLib_LOCK_TIMEOUT</b> |
| <b>SEE ALSO</b>     | <b>semSmLib</b> , <b>semLib</b> , <b>semBLib</b> , <b>smObjLib</b> , <b>semShow</b> , <i>VxWorks Programmer's Guide: Basic OS</i>          |

---

## semCCreate()

**NAME** **semCCreate()** – create and initialize a counting semaphore

**SYNOPSIS**

```
SEM_ID semCCreate  
(  
    int options,                /* semaphore option modes */  
    int initialCount           /* initial count */  
)
```

**DESCRIPTION** This routine allocates and initializes a counting semaphore. The semaphore is initialized to the specified initial count.

The *options* parameter specifies the queuing style for blocked tasks. Tasks may be queued on a priority basis or a first-in-first-out basis. These options are **SEM\_Q\_PRIORITY** (0x1) and **SEM\_Q\_FIFO** (0x0), respectively. That parameter also specifies if **semGive()** should return **ERROR** when the semaphore fails to send events. This option is turned off by default; it is activated by doing a bitwise-OR of **SEM\_EVENTSEND\_ERR\_NOTIFY** (0x10) with the queuing style of the semaphore.

**RETURNS** The semaphore ID, or **NULL** if memory cannot be allocated.

**SEE ALSO** **semCLib**

---

## **semClear()**

**NAME** **semClear()** – take a release 4.x semaphore, if the semaphore is available

**SYNOPSIS**

```
STATUS semClear  
(  
    SEM_ID semId           /* semaphore ID to empty */  
)
```

**DESCRIPTION** This routine takes a VxWorks 4.x semaphore if it is available (full), otherwise no action is taken except to return **ERROR**. This routine never preempts the caller.

**RETURNS** **OK**, or **ERROR** if the semaphore is unavailable.

**SEE ALSO** **semOLib**

---

## **semCreate()**

**NAME** **semCreate()** – create and initialize a release 4.x binary semaphore

**SYNOPSIS** **SEM\_ID semCreate (void)**

**DESCRIPTION** This routine allocates a VxWorks 4.x binary semaphore. The semaphore is initialized to empty. After initialization, it must be given before it can be taken.

**RETURNS** The semaphore ID, or **NULL** if memory cannot be allocated.

**SEE ALSO** **semOLib, semInit()**

---

## semCSmCreate()

|                     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
|---------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>         | <b>semCSmCreate()</b> – create and initialize a shared memory counting semaphore (VxMP Opt.)                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| <b>SYNOPSIS</b>     | <pre><b>SEM_ID</b> semCSmCreate (     int options,           /* semaphore options */     int initialCount      /* initial semaphore count */ )</pre>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| <b>DESCRIPTION</b>  | <p>This routine allocates and initializes a shared memory counting semaphore. The initial count value of the semaphore is specified by <i>initialCount</i>.</p> <p>The semaphore ID returned by this routine can be used directly by the generic semaphore-handling routines in <b>semLib</b> -- <b>semGive()</b>, <b>semTake()</b> and <b>semFlush()</b> -- and the show routines, such as <b>show()</b> and <b>semShow()</b>.</p> <p>The queuing style for blocked tasks is set by <i>options</i>; the only supported queuing style for shared memory semaphores is first-in-first-out, selected by <b>SEM_Q_FIFO</b>.</p> <p>Before this routine can be called, the shared memory objects facility must be initialized (see <b>semSmLib</b>).</p> <p>The maximum number of shared memory semaphores (binary plus counting) that can be created is <b>SM_OBJ_MAX_SEM</b>, a configurable parameter.</p> |
| <b>AVAILABILITY</b> | This routine is distributed as a component of the unbundled shared memory support option, VxMP.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| <b>RETURNS</b>      | The semaphore ID, or <b>NULL</b> if memory cannot be allocated from the shared semaphore dedicated memory partition.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| <b>ERRNO</b>        | <b>S_memLib_NOT_ENOUGH_MEMORY</b> , <b>S_semLib_INVALID_QUEUE_TYPE</b> ,<br><b>S_smObjLib_LOCK_TIMEOUT</b>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
| <b>SEE ALSO</b>     | <b>semSmLib</b> , <b>semLib</b> , <b>semCLib</b> , <b>smObjLib</b> , <b>semShow</b> , <i>VxWorks Programmer's Guide: Basic OS</i>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |

---

## semDelete()

**NAME** semDelete() – delete a semaphore

**SYNOPSIS**

```
STATUS semDelete
(
    SEM_ID semId          /* semaphore ID to delete */
)
```

**DESCRIPTION** This routine terminates and deallocates any memory associated with a specified semaphore. All tasks pending on the semaphore or pending for the reception of events meant to be sent from the semaphore will unblock and return **ERROR**.

---

**WARNING:** Take care when deleting semaphores, particularly those used for mutual exclusion, to avoid deleting a semaphore out from under a task that already has taken (owns) that semaphore. Applications should adopt the protocol of only deleting semaphores that the deleting task has successfully taken.

---

**RETURNS** OK, or **ERROR** if the semaphore ID is invalid.

**ERRNO**

**S\_intLib\_NOT\_ISR\_CALLABLE**  
Routine cannot be called from ISR.

**S\_objLib\_OBJ\_ID\_ERROR**  
Semaphore ID is invalid.

**S\_smObjLib\_NO\_OBJECT\_DESTROY**  
Deleting a shared semaphore is not permitted

**SEE ALSO** semLib, semBLib, semCLib, semMLib, semSmLib

---

## semEvStart()

**NAME** semEvStart() – start event notification process for a semaphore

**SYNOPSIS**

```
STATUS semEvStart
(
    SEM_ID semId,          /* semaphore on which to register events */
    UINT32 events,        /* 32 possible events to register */
    UINT8 options          /* event-related semaphore options */
)
```

**semEvStart()**

**DESCRIPTION** This routine turns on the event notification process for a given semaphore. When the semaphore becomes available but no task is pending on it, the events specified will be sent to the task registered by this function. A task can overwrite its own registration without first invoking **semEvStop()** or specifying the **ALLOW\_OVERWRITE** option.

The *option* parameter is used for 3 user options:

**EVENTS\_SEND\_ONCE** (0x1)

tells the semaphore to send the events one time only. Specify if the events are to be sent only once or every time the semaphore is free until **semEvStop()** is called.

**EVENTS\_ALLOW\_OVERWRITE** (0x2)

allows subsequent registrations to overwrite the current one. Specify if another task can register itself while the current task is still registered. If so, the current task registration is overwritten without any warning.

**EVENTS\_SEND\_IF\_FREE** (0x4)

tells the registration process to send events if the semaphore is free. Specify if events are to be sent at the time of the registration in the case the semaphore is free.

If none of these options are to be used, the option

**EVENTS\_OPTIONS\_NONE**

has to be passed to the *options* parameter.

---

**WARNING:** This routine cannot be called from interrupt level.

---

**RETURNS** OK on success, or **ERROR**.

**ERRNO****S\_objLib\_OBJ\_ID\_ERROR**

The semaphore ID is invalid.

**S\_eventLib\_ALREADY\_REGISTERED**

A task is already registered on the semaphore.

**S\_intLib\_NOT\_ISR\_CALLABLE**

Routine has been called from interrupt level.

**S\_eventLib\_EVENTSEND\_FAILED**

User chose to send events right away and that operation failed.

**S\_eventLib\_ZERO\_EVENTS**

User passed in a value of zero to the *events* parameter.

**SEE ALSO**

**semEvLib**, **eventLib**, **semLib**, **semEvStop()**

---

## semEvStop()

|                    |                                                                                                                                                                                                                                                 |
|--------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>        | <b>semEvStop()</b> – stop event notification process for a semaphore                                                                                                                                                                            |
| <b>SYNOPSIS</b>    | <pre><b>STATUS</b> semEvStop (     <b>SEM_ID</b> semId )</pre>                                                                                                                                                                                  |
| <b>DESCRIPTION</b> | This routine turns off the event notification process for a given semaphore. It thus allows another task to register itself for event notification on that particular semaphore.                                                                |
| <b>RETURNS</b>     | OK on success, or <b>ERROR</b> .                                                                                                                                                                                                                |
| <b>ERRNO</b>       | <b>S_objLib_OBJ_ID_ERROR</b><br>The semaphore ID is invalid.<br><b>S_intLib_NOT_ISR_CALLABLE</b><br>Routine has been called at interrupt level.<br><b>S_eventLib_TASK_NOT_REGISTERED</b><br>Routine has not been called by the registered task. |
| <b>SEE ALSO</b>    | <b>semEvLib, eventLib, semLib, semEvStart()</b>                                                                                                                                                                                                 |

---

## semFlush()

|                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
|--------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>        | <b>semFlush()</b> – unblock every task pending on a semaphore                                                                                                                                                                                                                                                                                                                                                                                                                     |
| <b>SYNOPSIS</b>    | <pre><b>STATUS</b> semFlush (     <b>SEM_ID</b> semId          /* semaphore ID to unblock everyone for */ )</pre>                                                                                                                                                                                                                                                                                                                                                                 |
| <b>DESCRIPTION</b> | <p>This routine atomically unblocks all tasks pending on a specified semaphore, <i>i.e.</i>, all tasks will be unblocked before any is allowed to run. The state of the underlying semaphore is unchanged. All pending tasks will enter the ready queue before having a chance to execute.</p> <p>The flush operation is useful as a means of broadcast in synchronization applications. Its use is illegal for mutual-exclusion semaphores created with <b>semMCreate()</b>.</p> |

**RETURNS** OK, or **ERROR** if the semaphore ID is invalid or the operation is not supported.

**ERRNO** **S\_objLib\_OBJ\_ID\_ERROR**

**SEE ALSO** **semLib, semBLib, semCLib, semMLib, semSmLib**

---

## semGive()

**NAME** **semGive()** – give a semaphore

**SYNOPSIS**

```
STATUS semGive  
(  
    SEM_ID semId          /* semaphore ID to give */  
)
```

**DESCRIPTION** This routine performs the give operation on a specified semaphore. Depending on the type of semaphore, the state of the semaphore and of the pending tasks may be affected. If no tasks are pending on the semaphore and a task has previously registered to receive events from the semaphore, these events are sent in the context of this call. This may result in the unpending of the task waiting for the events. If the semaphore fails to send events and if it was created using the **SEM\_EVENTSEND\_ERR\_NOTIFY** option, **ERROR** is returned even though the give operation was successful. The behavior of **semGive()** is discussed fully in the library description of the specific semaphore type being used.

**RETURNS** **OK** on success or **ERROR** otherwise

**ERRNO** **S\_intLib\_NOT\_ISR\_CALLABLE**  
Routine was called from an ISR for a mutex semaphore.

**S\_objLib\_OBJ\_ID\_ERROR**  
Semaphore ID is invalid.

**S\_semLib\_INVALID\_OPERATION**  
Current task not owner of mutex semaphore.

**S\_eventLib\_EVENTSEND\_FAILED**  
Semaphore failed to send events to the registered task. This **errno** value can only exist if the semaphore was created with the **SEM\_EVENTSEND\_ERR\_NOTIFY** option.

**SEE ALSO** **semLib, semBLib, semCLib, semMLib, semSmLib, semEvStart()**

---

## semInfo()

**NAME** `semInfo()` – get a list of task IDs that are blocked on a semaphore

**SYNOPSIS**

```
int semInfo
(
    SEM_ID semId,          /* semaphore ID to summarize */
    int idList[],         /* array of task IDs to be filled in */
    int maxTasks          /* max tasks idList can accommodate */
)
```

**DESCRIPTION** This routine reports the tasks blocked on a specified semaphore. Up to *maxTasks* task IDs are copied to the array specified by *idList*. The array is unordered.

---

**WARNING:** There is no guarantee that all listed tasks are still valid or that new tasks have not been blocked by the time `semInfo()` returns.

---

**RETURNS** The number of blocked tasks placed in *idList*.

**SEE ALSO** `semShow`

---

## semInit()

**NAME** `semInit()` – initialize a static binary semaphore

**SYNOPSIS**

```
STATUS semInit
(
    SEMAPHORE * pSemaphore /* 4.x semaphore to initialize */
)
```

**DESCRIPTION** This routine initializes static VxWorks 4.x semaphores. In some instances, a semaphore cannot be created with `semCreate()` but is a static object.

**RETURNS** OK, or ERROR if the semaphore cannot be initialized.

**SEE ALSO** `semOLib`, `semCreate()`

---

## semMCreate()

|                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
|--------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>        | <b>semMCreate()</b> – create and initialize a mutual-exclusion semaphore                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
| <b>SYNOPSIS</b>    | <pre>SEM_ID semMCreate (     int options          /* mutex semaphore options */ )</pre>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| <b>DESCRIPTION</b> | <p>This routine allocates and initializes a mutual-exclusion semaphore. The semaphore state is initialized to full.</p> <p>Semaphore options include the following:</p> <p><b>SEM_Q_PRIORITY</b> (0x1)<br/>Queue pended tasks on the basis of their priority.</p> <p><b>SEM_Q_FIFO</b> (0x0)<br/>Queue pended tasks on a first-in-first-out basis.</p> <p><b>SEM_DELETE_SAFE</b> (0x4)<br/>Protect a task that owns the semaphore from unexpected deletion. This option enables an implicit <b>taskSafe()</b> for each <b>semTake()</b>, and an implicit <b>taskUnsafe()</b> for each <b>semGive()</b>.</p> <p><b>SEM_INVERSION_SAFE</b> (0x8)<br/>Protect the system from priority inversion. With this option, the task owning the semaphore will execute at the highest priority of the tasks pended on the semaphore, if it is higher than its current priority. This option must be accompanied by the <b>SEM_Q_PRIORITY</b> queuing mode.</p> <p><b>SEM_EVENTSEND_ERR_NOTIFY</b> (0x10)<br/>When the semaphore is given, if a task is registered for events and the actual sending of events fails, a value of <b>ERROR</b> is returned and the <b>errno</b> is set accordingly. This option is off by default.</p> |
| <b>RETURNS</b>     | The semaphore ID, or <b>NULL</b> if the semaphore cannot be created.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| <b>ERRNO</b>       | <p><b>S_semLib_INVALID_OPTION</b><br/>Invalid option was passed to <b>semMCreate()</b>.</p> <p><b>S_memLib_NOT_ENOUGH_MEMORY</b><br/>Not enough memory available to create the semaphore.</p>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
| <b>SEE ALSO</b>    | <b>semMLib</b> , <b>semLib</b> , <b>semBLib</b> , <b>taskSafe()</b> , <b>taskUnsafe()</b>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |

---

## semMGiveForce()

**NAME** `semMGiveForce()` – give a mutual-exclusion semaphore without restrictions

**SYNOPSIS**

```
STATUS semMGiveForce  
(  
    SEM_ID semId          /* semaphore ID to give */  
)
```

**DESCRIPTION** This routine gives a mutual-exclusion semaphore, regardless of semaphore ownership. It is intended as a debugging aid only.

The routine is particularly useful when a task dies while holding some mutual-exclusion semaphore, because the semaphore can be resurrected. The routine will give the semaphore to the next task in the pend queue or make the semaphore full if no tasks are pending. In effect, execution will continue as if the task owning the semaphore had actually given the semaphore.

---

**WARNING:** This routine should be used only as a debugging aid, when the condition of the semaphore is known.

---

**RETURNS** OK, or **ERROR** if the semaphore ID is invalid.

**SEE ALSO** `semMLib`, `semGive()`

---

## semPxlLibInit()

**NAME** `semPxlLibInit()` – initialize POSIX semaphore support

**SYNOPSIS**

```
STATUS semPxlLibInit (void)
```

**DESCRIPTION** This routine must be called before using POSIX semaphores.

**RETURNS** OK, or **ERROR** if there is an error installing the semaphore library.

**SEE ALSO** `semPxlLib`

---

## semPxShowInit()

- NAME** `semPxShowInit()` – initialize the POSIX semaphore show facility
- SYNOPSIS** `STATUS semPxShowInit (void)`
- DESCRIPTION** This routine links the POSIX semaphore show routine into the VxWorks system. It is called automatically when the this show facility is configured into VxWorks using either of the following methods:
- If you use the configuration header files, define `INCLUDE_SHOW_ROUTINES` in `config.h`.
  - If you use the Tornado project facility, select `INCLUDE_POSIX_SEM_SHOW`.
- RETURNS** `OK`, or `ERROR` if an error occurs installing the file pointer show routine.
- SEE ALSO** `semPxShow`

---

## semShow()

- NAME** `semShow()` – show information about a semaphore
- SYNOPSIS** `STATUS semShow`
- ```
(  
    SEM_ID semId,          /* semaphore to display */  
    int level             /* 0 = summary, 1 = details */  
)
```
- DESCRIPTION** This routine displays the state and optionally the pended tasks of a semaphore. A summary of the state of the semaphore is displayed as follows:
- ```
Semaphore Id      : 0x585f2  
Semaphore Type    : BINARY  
Task Queuing      : PRIORITY  
Pended Tasks      : 1  
State             : EMPTY {Count if COUNTING, Owner if MUTEX}  
Options           : 0x1      SEM_Q_PRIORITY  
VxWorks Events  
-----  
Registered Task   : 0x594f0 (t1)  
Event(s) to Send  : 0x1
```

```
Options           : 0x7           EVENTS_SEND_ONCE
                  EVENTS_ALLOW_OVERWRITE
                  EVENTS_SEND_IF_FREE
```

If *level* is 1, then more detailed information will be displayed. If tasks are blocked on the queue, they are displayed in the order in which they will unblock, as follows:

```
Pended Tasks
-----
      NAME      TID      PRI  DELAY
-----
tExcTask      3fd678      0    21
tLogTask      3f8ac0      0   611
```

**RETURNS** OK or ERROR.

**SEE ALSO** *semShow*, *VxWorks Programmer's Guide: Target Shell*, *windsh*, *Tornado User's Guide: Shell*

## semShowInit()

**NAME** *semShowInit()* – initialize the semaphore show facility

**SYNOPSIS** `void semShowInit (void)`

**DESCRIPTION** This routine links the semaphore show facility into the VxWorks system. It is called automatically when the semaphore show facility is configured into VxWorks using either of the following methods:

- If you use the configuration header files, define `INCLUDE_SHOW_ROUTINES` in `config.h`.
- If you use the Tornado project facility, select `INCLUDE_SEM_SHOW`.

**RETURNS** N/A

**SEE ALSO** *semShow*

## semTake()

**NAME** semTake() – take a semaphore

**SYNOPSIS**

```
STATUS semTake
(
    SEM_ID semId,          /* semaphore ID to take */
    int timeout            /* timeout in ticks */
)
```

**DESCRIPTION** This routine performs the take operation on a specified semaphore. Depending on the type of semaphore, the state of the semaphore and the calling task may be affected. The behavior of **semTake()** is discussed fully in the library description of the specific semaphore type being used.

A timeout in ticks may be specified. If a task times out, **semTake()** will return **ERROR**. Timeouts of **WAIT\_FOREVER** (-1) and **NO\_WAIT** (0) indicate to wait indefinitely or not to wait at all.

When **semTake()** returns due to timeout, it sets the **errno** to **S\_objLib\_OBJ\_TIMEOUT** (defined in **objLib.h**).

The **semTake()** routine is not callable from interrupt service routines.

**RETURNS** OK, or **ERROR** if the semaphore ID is invalid or the task timed out.

**ERRNO** S\_intLib\_NOT\_ISR\_CALLABLE, S\_objLib\_OBJ\_ID\_ERROR, S\_objLib\_OBJ\_UNAVAILABLE

**SEE ALSO** semLib, semBLib, semCLib, semMLib, semSmLib

---

## sem\_close()

**NAME** sem\_close() – close a named semaphore (POSIX)

**SYNOPSIS**

```
int sem_close
(
    sem_t * sem            /* semaphore descriptor */
)
```

**DESCRIPTION** This routine is called to indicate that the calling task is finished with the specified named semaphore, *sem*. Do not call this routine with an unnamed semaphore (*i.e.*, one created by

**sem\_init()**; the effects are undefined. The **sem\_close()** call deallocates any system resources allocated by the system for use by this task for this semaphore.

If the semaphore has not been removed with a call to **sem\_unlink()**, then **sem\_close()** has no effect on the state of the semaphore. However, if the semaphore has been unlinked, the semaphore vanishes when the last task closes it.

---

**WARNING:** Take care to avoid risking the deletion of a semaphore that another task has already locked. Applications should only close semaphores that the closing task has opened.

---

**RETURNS** 0 (OK), or -1 (ERROR) if unsuccessful.

**ERRNO** EINVAL  
- invalid semaphore descriptor.

**SEE ALSO** semPxBLib, sem\_unlink(), sem\_open(), sem\_init()

---

## sem\_destroy()

**NAME** sem\_destroy() – destroy an unnamed semaphore (POSIX)

**SYNOPSIS**

```
int sem_destroy
(
    sem_t * sem          /* semaphore descriptor */
)
```

**DESCRIPTION** This routine is used to destroy the unnamed semaphore indicated by *sem*.

The **sem\_destroy()** call can only destroy a semaphore created by **sem\_init()**. Calling **sem\_destroy()** with a named semaphore will cause a **EINVAL** error. Subsequent use of the *sem* semaphore will cause an **EINVAL** error in the calling function.

If one or more tasks is blocked on the semaphore, the semaphore is not destroyed.

---

**WARNING:** Take care when deleting semaphores, particularly those used for mutual exclusion, to avoid deleting a semaphore out from under a task that has already locked that semaphore. Applications should adopt the protocol of only deleting semaphores that the deleting task has successfully locked.

---

**RETURNS** 0 (OK), or -1 (ERROR) if unsuccessful.

**sem\_getvalue()**

**ERRNO**           EINVAL  
                   - invalid semaphore descriptor.

**EBUSY**  
                   - one or more tasks is blocked on the semaphore.

**SEE ALSO**       semPxBLib, sem\_init()

---

## sem\_getvalue()

**NAME**           sem\_getvalue() – get the value of a semaphore (POSIX)

**SYNOPSIS**

```
int sem_getvalue
(
    sem_t * sem,           /* semaphore descriptor */
    int *  sval           /* buffer by which the value is returned */
)
```

**DESCRIPTION**   This routine updates the location referenced by the *sval* argument to have the value of the semaphore referenced by *sem* without affecting the state of the semaphore. The updated value represents an actual semaphore value that occurred at some unspecified time during the call, but may not be the actual value of the semaphore when it is returned to the calling task.

If *sem* is locked, the value returned by **sem\_getvalue()** will either be zero or a negative number whose absolute value represents the number of tasks waiting for the semaphore at some unspecified time during the call.

**RETURNS**       0 (OK), or -1 (ERROR) if unsuccessful.

**ERRNO**           EINVAL  
                   - invalid semaphore descriptor.

**SEE ALSO**       semPxBLib, sem\_post(), sem\_trywait(), sem\_trywait()

---

## sem\_init()

|                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
|--------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>        | <code>sem_init()</code> – initialize an unnamed semaphore (POSIX)                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| <b>SYNOPSIS</b>    | <pre>int sem_init (     sem_t *    sem,          /* semaphore to be initialized */     int        pshared,     /* process sharing */     unsigned int value      /* semaphore initialization value */ )</pre>                                                                                                                                                                                                                                                                                                             |
| <b>DESCRIPTION</b> | <p>This routine is used to initialize the unnamed semaphore <i>sem</i>. The value of the initialized semaphore is <i>value</i>. Following a successful call to <code>sem_init()</code> the semaphore may be used in subsequent calls to <code>sem_wait()</code>, <code>sem_trywait()</code>, and <code>sem_post()</code>. This semaphore remains usable until the semaphore is destroyed.</p> <p>The <i>pshared</i> parameter currently has no effect.</p> <p>Only <i>sem</i> itself may be used for synchronization.</p> |
| <b>RETURNS</b>     | 0 (OK), or -1 (ERROR) if unsuccessful.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| <b>ERRNO</b>       | <code>EINVAL</code><br>- <i>value</i> exceeds <code>SEM_VALUE_MAX</code> .<br><code>ENOSPC</code><br>- unable to initialize semaphore due to resource constraints.                                                                                                                                                                                                                                                                                                                                                        |
| <b>SEE ALSO</b>    | <code>semPxBLib</code> , <code>sem_wait()</code> , <code>sem_trywait()</code> , <code>sem_post()</code>                                                                                                                                                                                                                                                                                                                                                                                                                   |

---

## sem\_open()

|                 |                                                                                                                                                                                                 |
|-----------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>     | <code>sem_open()</code> – initialize/open a named semaphore (POSIX)                                                                                                                             |
| <b>SYNOPSIS</b> | <pre>sem_t * sem_open (     const char * name,      /* semaphore name */     int        oflag,      /* semaphore creation flags */     ...              /* extra optional parameters */ )</pre> |

**sem\_open()**

**DESCRIPTION** This routine establishes a connection between a named semaphore and a task. Following a call to **sem\_open()** with a semaphore name *name*, the task may reference the semaphore associated with *name* using the address returned by this call. This semaphore may be used in subsequent calls to **sem\_wait()**, **sem\_trywait()**, and **sem\_post()**. The semaphore remains usable until the semaphore is closed by a successful call to **sem\_close()**.

The *oflag* argument controls whether the semaphore is created or merely accessed by the call to **sem\_open()**. The following flag bits may be set in *oflag*:

**O\_CREAT**

Use this flag to create a semaphore if it does not already exist. If **O\_CREAT** is set and the semaphore already exists, **O\_CREAT** has no effect except as noted below under **O\_EXCL**. Otherwise, **sem\_open()** creates a semaphore. **O\_CREAT** requires a third and fourth argument: *mode*, which is of type **mode\_t**, and *value*, which is of type unsigned int. *mode* has no effect in this implementation. The semaphore is created with an initial value of *value*. Valid initial values for semaphores must be less than or equal to **SEM\_VALUE\_MAX**.

**O\_EXCL**

If **O\_EXCL** and **O\_CREAT** are set, **sem\_open()** will fail if the semaphore name exists. If **O\_EXCL** is set and **O\_CREAT** is not set, the named semaphore is not created.

To determine whether a named semaphore already exists in the system, call **sem\_open()** with the flags **O\_CREAT | O\_EXCL**. If the **sem\_open()** call fails, the semaphore exists.

If a task makes multiple calls to **sem\_open()** with the same value for *name*, then the same semaphore address is returned for each such call, provided that there have been no calls to **sem\_unlink()** for this semaphore.

References to copies of the semaphore will produce undefined results.

**NOTE** The current implementation has the following limitations:

- A semaphore cannot be closed with calls to **\_exit()** or **exec()**.
- A semaphore cannot be implemented as a file.
- Semaphore names will not appear in the file system.

**RETURNS** A pointer to **sem\_t**, or -1 (**ERROR**) if unsuccessful.

**ERRNO****EEXIST**

- **O\_CREAT | O\_EXCL** are set and the semaphore already exists.

**EINVAL**

- *value* exceeds **SEM\_VALUE\_MAX** or the semaphore name is invalid.

**ENAMETOOLONG**

- the semaphore name is too long.

**ENOENT**

- the named semaphore does not exist and **O\_CREAT** is not set.

ENOSPC

- the semaphore could not be initialized due to resource constraints.

SEE ALSO **semPxBLib, sem\_unlink()**

---

## sem\_post()

NAME **sem\_post()** – unlock (give) a semaphore (POSIX)

SYNOPSIS

```
int sem_post
(
    sem_t * sem          /* semaphore descriptor */
)
```

DESCRIPTION This routine unlocks the semaphore referenced by *sem* by performing the semaphore unlock operation on that semaphore.

If the semaphore value resulting from the operation is positive, then no tasks were blocked waiting for the semaphore to become unlocked; the semaphore value is simply incremented.

If the value of the semaphore resulting from this semaphore is zero, then one of the tasks blocked waiting for the semaphore will return successfully from its call to **sem\_wait()**.

---

**NOTE:** The `_POSIX_PRIORITY_SCHEDULING` functionality is not yet supported.

---

Note that the POSIX terms *unlock* and *post* correspond to the term *give* used in other VxWorks semaphore documentation.

RETURNS 0 (OK), or -1 (ERROR) if unsuccessful.

ERRNO **EINVAL**  
- invalid semaphore descriptor.

SEE ALSO **semPxBLib, sem\_wait(), sem\_trywait()**

## sem\_trywait()

**NAME** sem\_trywait() – lock (take) a semaphore, returning error if unavailable (POSIX)

**SYNOPSIS**

```
int sem_trywait
(
    sem_t * sem          /* semaphore descriptor */
)
```

**DESCRIPTION** This routine locks the semaphore referenced by *sem* only if the semaphore is currently not locked; that is, if the semaphore value is currently positive. Otherwise, it does not lock the semaphore. In either case, this call returns immediately without blocking.

Upon return, the state of the semaphore is always locked (either as a result of this call or by a previous **sem\_wait()** or **sem\_trywait()**). The semaphore will remain locked until **sem\_post()** is executed and returns successfully.

Deadlock detection is not implemented.

Note that the POSIX term *lock* corresponds to the term *take* used in other VxWorks semaphore documentation.

**RETURNS** 0 (OK), or -1 (ERROR) if unsuccessful.

**ERRNO** EAGAIN  
- semaphore is already locked.  
EINVAL  
- invalid semaphore descriptor.

**SEE ALSO** semPxBLib, sem\_wait(), sem\_post()

---

## sem\_unlink()

**NAME** sem\_unlink() – remove a named semaphore (POSIX)

**SYNOPSIS**

```
int sem_unlink
(
    const char * name    /* semaphore name */
)
```

|                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
|--------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>DESCRIPTION</b> | This routine removes the string <i>name</i> from the semaphore name table, and marks the corresponding semaphore for destruction. An unlinked semaphore is destroyed when the last task closes it with <b>sem_close()</b> . After a particular name is removed from the table, calls to <b>sem_open()</b> using the same name cannot connect to the same semaphore, even if other tasks are still using it. Instead, such calls refer to a new semaphore with the same name. |
| <b>RETURNS</b>     | 0 (OK), or -1 (ERROR) if unsuccessful.                                                                                                                                                                                                                                                                                                                                                                                                                                       |
| <b>ERRNO</b>       | <b>ENAMETOOLONG</b><br>- semaphore name too long.<br><b>ENOENT</b><br>- named semaphore does not exist.                                                                                                                                                                                                                                                                                                                                                                      |
| <b>SEE ALSO</b>    | <b>semPxBLib</b> , <b>sem_open()</b> , <b>sem_close()</b>                                                                                                                                                                                                                                                                                                                                                                                                                    |

---

## sem\_wait()

|                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
|--------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>        | <b>sem_wait()</b> – lock (take) a semaphore, blocking if not available (POSIX)                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
| <b>SYNOPSIS</b>    | <pre>int sem_wait (     sem_t * sem          /* semaphore descriptor */ )</pre>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| <b>DESCRIPTION</b> | <p>This routine locks the semaphore referenced by <i>sem</i> by performing the semaphore lock operation on that semaphore. If the semaphore value is currently zero, the calling task will not return from the call to <b>sem_wait()</b> until it either locks the semaphore or the call is interrupted by a signal.</p> <p>On return, the state of the semaphore is locked and will remain locked until <b>sem_post()</b> is executed and returns successfully.</p> <p>Deadlock detection is not implemented.</p> <p>Note that the POSIX term <i>lock</i> corresponds to the term <i>take</i> used in other VxWorks documentation regarding semaphores.</p> |
| <b>RETURNS</b>     | 0 (OK), or -1 (ERROR) if unsuccessful.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
| <b>ERRNO</b>       | <b>EINVAL</b><br>- invalid semaphore descriptor, or semaphore destroyed while task waiting.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
| <b>SEE ALSO</b>    | <b>semPxBLib</b> , <b>sem_trywait()</b> , <b>sem_post()</b>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |

**send()**

---

|                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
|--------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>        | <b>send()</b> – send data to a socket                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| <b>SYNOPSIS</b>    | <pre>int send (     int          s,          /* socket to send to */     const char * buf,       /* pointer to buffer to transmit */     int          buflen,    /* length of buffer */     int          flags      /* flags to underlying protocols */ )</pre>                                                                                                                                                                                                                                                                                         |
| <b>DESCRIPTION</b> | <p>This routine transmits data to a previously established connection-based (stream) socket. The maximum length of <i>buf</i> is subject to the limits on TCP buffer size; see the discussion of <code>SO_SNDBUF</code> in the <code>setsockopt()</code> manual entry.</p> <p>You may OR the following values into the <i>flags</i> parameter with this operation:</p> <ul style="list-style-type: none"><li><code>MSG_OOB</code> (0x1)<br/>Out-of-band data.</li><li><code>MSG_DONTROUTE</code> (0x4)<br/>Send without using routing tables.</li></ul> |
| <b>RETURNS</b>     | The number of bytes sent, or <code>ERROR</code> if the call fails.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| <b>SEE ALSO</b>    | <code>sockLib</code> , <code>setsockopt()</code> , <code>sendmsg()</code>                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |

---

**sendAdvert()**

|                 |                                                                                   |
|-----------------|-----------------------------------------------------------------------------------|
| <b>NAME</b>     | <b>sendAdvert()</b> – send an advertisement to one location                       |
| <b>SYNOPSIS</b> | <pre>void sendAdvert (     int          index,     struct in_addr dstAddr )</pre> |

|                    |                                                                                                                                                                                               |
|--------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>DESCRIPTION</b> | This routine sends a router advertisement using the data stored for its' corresponding interface. Only the primary network address of the interface is used as the advertised router address. |
| <b>RETURNS</b>     | N/A                                                                                                                                                                                           |
| <b>SEE ALSO</b>    | <b>rdiscLib</b>                                                                                                                                                                               |

---

## sendAdvertAll()

|                    |                                                                                                                                                                                               |
|--------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>        | <b>sendAdvertAll()</b> – send an advertisement to all active locations                                                                                                                        |
| <b>SYNOPSIS</b>    | <b>void sendAdvertAll (void)</b>                                                                                                                                                              |
| <b>DESCRIPTION</b> | This routine sends a router advertisement using the data stored for each corresponding interface. Only the primary network address of the interface is used as the advertised router address. |
| <b>RETURNS</b>     | N/A                                                                                                                                                                                           |
| <b>SEE ALSO</b>    | <b>rdiscLib</b>                                                                                                                                                                               |

---

## sendmsg()

|                    |                                                                                                                                                                                                                    |
|--------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>        | <b>sendmsg()</b> – send a message to a socket                                                                                                                                                                      |
| <b>SYNOPSIS</b>    | <pre>int sendmsg (     int          sd,          /* socket to send to */     struct msghdr * mp,      /* scatter-gather message header */     int          flags       /* flags to underlying protocols */ )</pre> |
| <b>DESCRIPTION</b> | This routine sends a message to a datagram socket. It may be used in place of <b>sendto()</b> to decrease the overhead of reconstructing the message-header structure ( <b>msghdr</b> ) for each message.          |

**sendto()**

For BSD 4.4 sockets a copy of the *mp>msg\_iov* array will be made. This requires a cluster from the network stack system pool of size *mp>msg\_iovlen \* sizeof (struct iovec)* or 8 bytes.

**RETURNS** The number of bytes sent, or **ERROR** if the call fails.

**SEE ALSO** `sockLib`, `sendto()`

---

## sendto()

**NAME** `sendto()` – send a message to a socket

**SYNOPSIS**

```
int sendto
(
    int          s,          /* socket to send data to */
    caddr_t     buf,        /* pointer to data buffer */
    int         buflen,     /* length of buffer */
    int         flags,      /* flags to underlying protocols */
    struct sockaddr * to,   /* recipient's address */
    int         tolen      /* length of to sockaddr */
)
```

**DESCRIPTION** This routine sends a message to the datagram socket named by *to*. The socket *s* is received by the receiver as the sending socket.

The maximum length of *buf* is subject to the limits on UDP buffer size. See the discussion of `SO_SNDBUF` in the `setsockopt()` manual entry.

You can OR the following values into the *flags* parameter with this operation:

`MSG_OOB` (0x1)

Out-of-band data.

`MSG_DONTROUTE` (0x4)

Send without using routing tables.

**RETURNS** The number of bytes sent, or **ERROR** if the call fails.

**SEE ALSO** `sockLib`, `setsockopt()`

---

## set\_new\_handler()

|                      |                                                                                                                                                                                                                                             |
|----------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>          | <b>set_new_handler()</b> – set <b>new_handler</b> to user-defined function (C++)                                                                                                                                                            |
| <b>SYNOPSIS</b>      | <code>extern void (*set_new_handler (void(* pNewNewHandler)())) ()</code>                                                                                                                                                                   |
| <b>DESCRIPTION</b>   | <p>This function is used to define the function that will be called when operator new cannot allocate memory.</p> <p>The <b>new_handler</b> acts for all threads in the system; you cannot set a different handler for different tasks.</p> |
| <b>RETURNS</b>       | A pointer to the previous value of <b>new_handler</b> .                                                                                                                                                                                     |
| <b>INCLUDE FILES</b> | new                                                                                                                                                                                                                                         |
| <b>SEE ALSO</b>      | cplusLib                                                                                                                                                                                                                                    |

---

## set\_terminate()

|                      |                                                                                                                                                                                                                                                                |
|----------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>          | <b>set_terminate()</b> – set terminate to user-defined function (C++)                                                                                                                                                                                          |
| <b>SYNOPSIS</b>      | <code>extern void (*set_terminate (void(* terminate_handler)())) ()</code>                                                                                                                                                                                     |
| <b>DESCRIPTION</b>   | <p>This function is used to define the <b>terminate_handler</b> which will be called when an uncaught exception is raised.</p> <p>The <b>terminate_handler</b> acts for all threads in the system; you cannot set a different handler for different tasks.</p> |
| <b>RETURNS</b>       | The previous <b>terminate_handler</b> .                                                                                                                                                                                                                        |
| <b>INCLUDE FILES</b> | exception                                                                                                                                                                                                                                                      |
| <b>SEE ALSO</b>      | cplusLib                                                                                                                                                                                                                                                       |

**setbuf()**


---

|                      |                                                                                                                                                                                                                                                                      |
|----------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>          | <b>setbuf()</b> – specify the buffering for a stream (ANSI)                                                                                                                                                                                                          |
| <b>SYNOPSIS</b>      | <pre> void setbuf (     FILE * fp,                /* stream to set buffering for */     char * buf                /* buffer to use */ ) </pre>                                                                                                                       |
| <b>DESCRIPTION</b>   | Except that it returns no value, this routine is equivalent to <b>setvbuf()</b> invoked with the <i>mode</i> <b>_IOFBF</b> (full buffering) and <i>size</i> <b>BUFSIZ</b> , or (if <i>buf</i> is a null pointer), with the <i>mode</i> <b>_IONBF</b> (no buffering). |
| <b>INCLUDE FILES</b> | <b>stdio.h</b>                                                                                                                                                                                                                                                       |
| <b>RETURNS</b>       | N/A                                                                                                                                                                                                                                                                  |
| <b>SEE ALSO</b>      | <b>ansiStdio, setvbuf()</b>                                                                                                                                                                                                                                          |

---

**setbuffer()**

|                      |                                                                                                                                                                                                                                                                                                                                                                                                                         |
|----------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>          | <b>setbuffer()</b> – specify buffering for a stream                                                                                                                                                                                                                                                                                                                                                                     |
| <b>SYNOPSIS</b>      | <pre> void setbuffer (     FILE * fp,                /* stream to set buffering for */     char * buf,              /* buffer to use */     int  size                 /* buffer size */ ) </pre>                                                                                                                                                                                                                        |
| <b>DESCRIPTION</b>   | <p>This routine specifies a buffer <i>buf</i> to be used for a stream in place of the automatically allocated buffer. If <i>buf</i> is <b>NULL</b>, the stream is unbuffered. This routine should be called only after the stream has been associated with an open file and before any other operation is performed on the stream.</p> <p>This routine is provided for compatibility with earlier VxWorks releases.</p> |
| <b>INCLUDE FILES</b> | <b>stdio.h</b>                                                                                                                                                                                                                                                                                                                                                                                                          |

**RETURNS** N/A

**SEE ALSO** `ansiStdio`, `setvbuf()`

---

## sethostname()

**NAME** `sethostname()` – set the symbolic name of this machine

**SYNOPSIS**

```
int sethostname
(
    char * name,          /* machine name */
    int  nameLen         /* length of name */
)
```

**DESCRIPTION** This routine sets the target machine's symbolic name, which can be used for identification.

**RETURNS** OK or ERROR.

**SEE ALSO** `hostLib`

---

## setjmp()

**NAME** `setjmp()` – save the calling environment in a `jmp_buf` argument (ANSI)

**SYNOPSIS**

```
int setjmp
(
    jmp_buf env
)
```

**DESCRIPTION** This routine saves the calling environment in `env`, in order to permit a `longjmp()` call to restore that environment (thus performing a non-local goto).

### Constraints on Calling Environment

The `setjmp()` routine may only be used in the following contexts:

- as the entire controlling expression of a selection or iteration statement;
- as one operand of a relational or equality operator, in the controlling expression of a selection or iteration statement;

## **setlinebuf()**

- as the operand of a single-argument **!** operator, in the controlling expression of a selection or iteration statement; or
- as a complete C statement containing nothing other than the **setjmp()** call (though the result may be cast to **void**)

**RETURNS** From a direct invocation, **setjmp()** returns zero. From a call to **longjmp()**, it returns a non-zero value specified as an argument to **longjmp()**.

**SEE ALSO** **ansiSetjmp**, **longjmp()**

---

## **setlinebuf()**

**NAME** **setlinebuf()** – set line buffering for standard output or standard error

**SYNOPSIS**

```
int setlinebuf
(
    FILE * fp                /* stream - stdout or stderr */
)
```

**DESCRIPTION** This routine changes **stdout** or **stderr** streams from block-buffered or unbuffered to line-buffered. Unlike **setbuf()**, **setbuffer()**, or **setvbuf()**, it can be used at any time the stream is active.

A stream can be changed from unbuffered or line-buffered to fully buffered using **freopen()**. A stream can be changed from fully buffered or line-buffered to unbuffered using **freopen()** followed by **setbuf()** with a buffer argument of **NULL**.

This routine is provided for compatibility with earlier VxWorks releases.

**INCLUDE** **stdio.h**

**RETURNS** **OK**, or **ERROR** if *fp* is not a valid stream.

**SEE ALSO** **ansiStdio**

---

## setlocale()

|                      |                                                                                                                                                                                                                                                                                                  |
|----------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>          | <b>setlocale()</b> – set the appropriate locale (ANSI)                                                                                                                                                                                                                                           |
| <b>SYNOPSIS</b>      | <pre>char *setlocale (     int          category,    /* category to change */     const char * localeName  /* locale name */ )</pre>                                                                                                                                                             |
| <b>DESCRIPTION</b>   | <p>This function is included for ANSI compatibility. Only the default is implemented. At program start-up, the equivalent of the following is executed:</p> <pre>setlocale (LC_ALL, "C");</pre> <p>This specifies the program's entire locale and the minimal environment for C translation.</p> |
| <b>INCLUDE FILES</b> | <b>locale.h, string.h, stdlib.h</b>                                                                                                                                                                                                                                                              |
| <b>RETURNS</b>       | A pointer to the string "C".                                                                                                                                                                                                                                                                     |
| <b>SEE ALSO</b>      | <b>ansiLocale</b>                                                                                                                                                                                                                                                                                |

---

## setsockopt()

|                    |                                                                                                                                                                                                                                                                                                          |
|--------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>        | <b>setsockopt()</b> – set socket options                                                                                                                                                                                                                                                                 |
| <b>SYNOPSIS</b>    | <pre>STATUS setsockopt (     int    s,                /* target socket */     int    level,            /* protocol level of option */     int    optname,          /* option name */     char * optval,           /* pointer to option value */     int    optlen            /* option length */ )</pre> |
| <b>DESCRIPTION</b> | <p>This routine sets the options associated with a socket. To manipulate options at the "socket" level, <i>level</i> should be <b>SOL_SOCKET</b>. Any other levels should use the appropriate protocol number.</p>                                                                                       |

## OPTIONS FOR STREAM SOCKETS

The following sections discuss the socket options available for stream (TCP) sockets.

### SO\_KEEPALIVE -- Detecting a Dead Connection

Specify the `SO_KEEPALIVE` option to make the transport protocol (TCP) initiate a timer to detect a dead connection:

```
setsockopt (sock, SOL_SOCKET, SO_KEEPALIVE, &optval, sizeof (optval));
```

This prevents an application from hanging on an invalid connection. The value at *optval* for this option is an integer (type `int`), either 1 (on) or 0 (off).

The integrity of a connection is verified by transmitting zero-length TCP segments triggered by a timer, to force a response from a peer node. If the peer does not respond after repeated transmissions of the `KEEPALIVE` segments, the connection is dropped, all protocol data structures are reclaimed, and processes sleeping on the connection are awakened with an `ETIMEDOUT` error.

The `ETIMEDOUT` timeout can happen in two ways. If the connection is not yet established, the `KEEPALIVE` timer expires after idling for `TCPTV_KEEP_INIT`. If the connection is established, the `KEEPALIVE` timer starts up when there is no traffic for `TCPTV_KEEP_IDLE`. If no response is received from the peer after sending the `KEEPALIVE` segment `TCPTV_KEEPCNT` times with interval `TCPTV_KEEPINTVL`, TCP assumes that the connection is invalid. The `TCPTV_KEEP_INIT`, `TCPTV_KEEP_IDLE`, `TCPTV_KEEPCNT`, and `TCPTV_KEEPINTVL` parameters are defined in the file `target/h/netinet/tcp_timer.h`.

### SO\_LINGER -- Closing a Connection

Specify the `SO_LINGER` option to determine whether TCP should perform a “graceful” close:

```
setsockopt (sock, SOL_SOCKET, SO_LINGER, &optval, sizeof (optval));
```

To achieve a “graceful” close in response to the shutdown of a connection, TCP puts itself through an elaborate set of state transitions. The goal is to assure that all the unacknowledged data in the transmission channel are acknowledged, and that the peer is shut down properly.

The value at *optval* indicates the amount of time to linger if there is unacknowledged data, using `struct linger` in `target/h/sys/socket.h`. The `linger` structure has two members: `l_onoff` and `l_linger`. `l_onoff` can be set to 1 to turn on the `SO_LINGER` option, or set to 0 to turn off the `SO_LINGER` option. `l_linger` indicates the amount of time to linger. If `l_onoff` is turned on and `l_linger` is set to 0, a default value `TCP_LINGERTIME` (specified in `netinet/tcp_timer.h`) is used for incoming connections accepted on the socket.

When `SO_LINGER` is turned on and the `l_linger` field is set to 0, TCP simply drops the connection by sending out an RST (if a connection is already established). This frees up the space for the TCP protocol control block, and wakes up all tasks sleeping on the socket.

For the client side socket, the value of **l\_linger** is not changed if it is set to 0. To make sure that the value of **l\_linger** is 0 on a newly accepted socket connection, issue another **setsockopt()** after the **accept()** call.

Currently the exact value of **l\_linger** time is actually ignored (other than checking for 0); that is, TCP performs the state transitions if **l\_linger** is not 0, but does not explicitly use its value.

#### **TCP\_NODELAY -- Delivering Messages Immediately**

Specify the **TCP\_NODELAY** option for real-time protocols, such as the X Window System Protocol, that require immediate delivery of many small messages:

```
setsockopt (sock, IPPROTO_TCP, TCP_NODELAY, &optval, sizeof (optval));
```

The value at *optval* is an integer (type **int**) set to either 1 (on) or 0 (off).

By default, the VxWorks TCP implementation employs an algorithm that attempts to avoid the congestion that can be produced by a large number of small TCP segments. This typically arises with virtual terminal applications (such as **telnet** or **rlogin**) across networks that have low bandwidth and long delays. The algorithm attempts to have no more than one outstanding unacknowledged segment in the transmission channel while queueing up the rest of the smaller segments for later transmission. Another segment is sent only if enough new data is available to make up a maximum sized segment, or if the outstanding data is acknowledged.

This congestion-avoidance algorithm works well for virtual terminal protocols and bulk data transfer protocols such as FTP without any noticeable side effects. However, real-time protocols that require immediate delivery of many small messages, such as the X Window System Protocol, need to defeat this facility to guarantee proper responsiveness in their operation.

**TCP\_NODELAY** is a mechanism to turn off the use of this algorithm. If this option is turned on and there is data to be sent out, TCP bypasses the congestion-avoidance algorithm: any available data segments are sent out if there is enough space in the send window.

#### **TCP\_MAXSEG -- Changing TCP MSS for the connection**

Specify the **TCP\_MAXSEG** option to decrease the maximum allowable size of an outgoing TCP segment. This option cannot be used to increase the MSS.

```
setsockopt (sock, IPPROTO_TCP, TCP_MAXSEG, &optval, sizeof (optval));
```

The value at *optval* is an integer set to the desired MSS (*e.g.*, 1024).

When a TCP socket is created, the MSS is initialized to the default MSS value which is determined by the configuration parameter **TCP\_MSS\_DFLT** (512 by default). When a connection request is received from the other end with an MSS option, the MSS is modified depending on the value of the received MSS and on the results of Path MTU Discovery (which is enabled by default). The MSS may be set as high as the outgoing interface MTU (1460 for an Ethernet). Therefore, after a call to **socket** but before a connection is established, an application can only decrease the MSS from its default of 512.

**setsockopt()**

After a connection is established, the application can decrease the MSS from whatever value was selected.

**SO\_DEBUG -- Debugging the underlying protocol**

Specify the `SO_DEBUG` option to let the underlying protocol module record debug information.

```
setsockopt (sock, SOL_SOCKET, SO_DEBUG, &optval, sizeof (optval));
```

The value at *optval* for this option is an integer (type `int`), either 1 (on) or 0 (off).

**OPTION FOR DATAGRAM SOCKETS**

The following section discusses an option for datagram (UDP) sockets.

**SO\_BROADCAST -- Sending to Multiple Destinations**

Specify the `SO_BROADCAST` option when an application needs to send data to more than one destination:

```
setsockopt (sock, SOL_SOCKET, SO_BROADCAST, &optval, sizeof (optval));
```

The value at *optval* is an integer (type `int`), either 1 (on) or 0 (off).

**OPTIONS FOR DATAGRAM AND RAW SOCKETS**

The following section discusses options for multicasting on UDP and RAW sockets.

**IP\_ADD\_MEMBERSHIP -- Join a Multicast Group**

Specify the `IP_ADD_MEMBERSHIP` option when a process needs to join multicast group:

```
setsockopt (sock, IPPROTO_IP, IP_ADD_MEMBERSHIP, (char *)&ipMreq,
           sizeof (ipMreq));
```

The value of *ipMreq* is an `ip_mreq` structure. `ipMreq.imr_multiaddr.s_addr` is the internet multicast address `ipMreq.imr_interface.s_addr` is the internet unicast address of the interface through which the multicast packet needs to pass.

**IP\_DROP\_MEMBERSHIP -- Leave a Multicast Group**

Specify the `IP_DROP_MEMBERSHIP` option when a process needs to leave a previously joined multicast group:

```
setsockopt (sock, IPPROTO_IP, IP_DROP_MEMBERSHIP, (char *)&ipMreq,
           sizeof (ipMreq));
```

The value of *ipMreq* is an `ip_mreq` structure. `ipMreq.imr_multiaddr.s_addr` is the internet multicast address. `ipMreq.imr_interface.s_addr` is the internet unicast address of the interface to which the multicast address was bound.

**IP\_MULTICAST\_IF -- Select a Default Interface for Outgoing Multicasts**

Specify the `IP_MULTICAST_IF` option when an application needs to specify an outgoing network interface through which all multicast packets are sent:

```
setsockopt (sock, IPPROTO_IP, IP_MULTICAST_IF, (char *)&ifAddr,
           sizeof (mCastAddr));
```

The value of *ifAddr* is an `in_addr` structure. `ifAddr.s_addr` is the internet network interface address.

#### IP\_MULTICAST\_TTL -- Select a Default TTL

Specify the `IP_MULTICAST_TTL` option when an application needs to select a default TTL (time to live) for outgoing multicast packets:

```
setsockopt (sock, IPPROTO_IP, IP_MULTICAST_TTL, &optval, sizeof(optval));
```

The value at *optval* is an integer (type `int`), time to live value.

| optval(TTL) | Application                   | Scope          |
|-------------|-------------------------------|----------------|
| 0           |                               | same interface |
| 1           |                               | same subnet    |
| 31          | local event video             |                |
| 32          |                               | same site      |
| 63          | local event audio             |                |
| 64          |                               | same region    |
| 95          | IETF channel 2 video          |                |
| 127         | IETF channel 1 video          |                |
| 128         |                               | same continent |
| 159         | IETF channel 2 audio          |                |
| 191         | IETF channel 1 audio          |                |
| 223         | IETF channel 2 low-rate audio |                |
| 255         | IETF channel 1 low-rate audio |                |
|             | unrestricted in scope         |                |

#### IP\_MULTICAST\_LOOP -- Enable or Disable Loopback

Enable or disable loopback of outgoing multicasts.

```
setsockopt (sock, IPPROTO_IP, IP_MULTICAST_LOOP, &optval, sizeof(optval));
```

The value at *optval* is an integer (type `int`), either 1(on) or 0 (off).

#### OPTIONS FOR DATAGRAM, STREAM AND RAW SOCKETS

The following section discusses options for RAW, DGRAM or STREAM sockets.

#### IP\_OPTIONS -- set options to be included in outgoing datagrams

Sets the IP options sent from this socket with every packet.

```
setsockopt (sock, IPPROTO_IP, IP_OPTIONS, optbuf, optbuflen);
```

Here *optbuf* is a buffer containing the options.

**IP\_TOS-- set options to be included in outgoing datagrams**

Sets the Type-Of-Service field for each packet sent from this socket.

```
setsockopt (sock, IPPROTO_IP, IP_TOS, &optval, sizeof(optval));
```

Here *optval* is an integer (type *int*). This integer can be set to `IP_TOS_LOWDELAY`, `IP_TOS_THROUGHPUT`, `IP_TOS_RELIABILITY`, or `IP_TOS_MINCOST`, to indicate how the packets sent on this socket should be prioritized.

**IP\_TTL-- set the time-to-live field in outgoing datagrams**

Sets the Time-To-Live field for each packet sent from this socket.

```
setsockopt (sock, IPPROTO_IP, IP_TTL, &optval, sizeof(optval));
```

Here *optval* is an integer (type *int*), indicating the number of hops a packet can take before it is discarded.

**IP\_RECVRTOPTS -- [un-]set queueing of reversed source route**

Sets whether or not reversed source route queueing will be enabled for incoming datagrams. (Not implemented)

```
setsockopt (sock, IPPROTO_IP, IP_RECVRTOPTS, &optval, sizeof(optval));
```

Here *optval* is a boolean (type *int*). However, this option is currently not implemented, so setting it will not change the behavior of the system.

**IP\_RECVDSTADDR -- [un-]set queueing of IP destination address**

Sets whether or not the socket will receive the IP address of the destination of an incoming datagram in control data.

```
setsockopt (sock, IPPROTO_IP, IP_RECVDSTADDR, &optval, sizeof(optval));
```

Here *optval* is a boolean (type *int*).

**OPTIONS FOR BOTH STREAM AND DATAGRAM SOCKETS**

The following sections describe options that can be used with either stream or datagram sockets.

**SO\_REUSEADDR -- Reusing a Socket Address**

Specify the `SO_REUSEADDR` option to bind a stream socket to a local port that may be still bound to another stream socket:

```
setsockopt (sock, SOL_SOCKET, SO_REUSEADDR, &optval, sizeof (optval));
```

The value at *optval* is an integer (type *int*), either 1 (on) or 0 (off).

When the `SO_REUSEADDR` option is turned on, applications may bind a stream socket to a local port. This is possible even if the port is still bound to another stream socket. It is even possible if that other socket is associated with a “zombie” protocol control block context that has not yet freed from previous sessions. The uniqueness of port number

combinations for each connection is still preserved through sanity checks performed at actual connection setup time. If this option is not turned on and an application attempts to bind to a port that is being used by a zombie protocol control block, the `bind()` call fails.

#### **SO\_REUSEPORT -- Reusing a Socket address and port**

This option is similar to the `SO_REUSEADDR` option but it allows binding to the same local address and port combination.

```
setsockopt (sock, SOL_SOCKET, SO_REUSEPORT, &optval, sizeof (optval));
```

The value at *optval* is an integer (type *int*), either 1 (on) or 0 (off).

The `SO_REUSEPORT` option is mainly required by multicast applications where a number of applications need to bind to the same multicast address and port to receive multicast data. Unlike `SO_REUSEADDR` where only the later applications need to set this option, with `SO_REUSEPORT` all applications including the first to bind to the port are required to set this option. For multicast addresses `SO_REUSEADDR` and `SO_REUSEPORT` show the same behavior so `SO_REUSEADDR` can be used instead.

#### **SO\_SNDBUF -- Specifying the Size of the Send Buffer**

Specify the `SO_SNDBUF` option to adjust the maximum size of the socket-level send buffer:

```
setsockopt (sock, SOL_SOCKET, SO_SNDBUF, &optval, sizeof (optval));
```

The value at *optval* is an integer (type *int*) that specifies the size of the socket-level send buffer to be allocated.

When stream or datagram sockets are created, each transport protocol reserves a set amount of space at the socket level for use when the sockets are attached to a protocol. For TCP, the default size of the send buffer is 8192 bytes. For UDP, the default size of the send buffer is 9216 bytes. Socket-level buffers are allocated dynamically from the mbuf pool.

The effect of setting the maximum size of buffers (for both `SO_SNDBUF` and `SO_RCVBUF`, described below) is not actually to allocate the mbufs from the mbuf pool. Instead, the effect is to set the high-water mark in the protocol data structure, which is used later to limit the amount of mbuf allocation. Thus, the maximum size specified for the socket level send and receive buffers can affect the performance of bulk data transfers. For example, the size of the TCP receive windows is limited by the remaining socket-level buffer space. These parameters must be adjusted to produce the optimal result for a given application.

#### **SO\_RCVBUF -- Specifying the Size of the Receive Buffer**

Specify the `SO_RCVBUF` option to adjust the maximum size of the socket-level receive buffer:

```
setsockopt (sock, SOL_SOCKET, SO_RCVBUF, &optval, sizeof (optval));
```

The value at *optval* is an integer (type *int*) that specifies the size of the socket-level receive buffer to be allocated.

**setsockopt()**

When stream or datagram sockets are created, each transport protocol reserves a set amount of space at the socket level for use when the sockets are attached to a protocol. For TCP, the default size is 8192 bytes. UDP reserves 41600 bytes, enough space for up to forty incoming datagrams (1 Kbyte each).

See the `SO_SNDBUF` discussion above for a discussion of the impact of buffer size on application performance.

**SO\_OOBINLINE -- Placing Urgent Data in the Normal Data Stream**

Specify the `SO_OOBINLINE` option to place urgent data within the normal receive data stream:

```
setsockopt (sock, SOL_SOCKET, SO_OOBINLINE, &optval, sizeof (optval));
```

TCP provides an expedited data service that does not conform to the normal constraints of sequencing and flow control of data streams. The expedited service delivers “out-of-band” (urgent) data ahead of other “normal” data to provide interrupt-like services (for example, when you hit a CTRL-C during `telnet` or `rlogin` session while data is being displayed on the screen.)

TCP does not actually maintain a separate stream to support the urgent data. Instead, urgent data delivery is implemented as a pointer (in the TCP header) which points to the sequence number of the octet following the urgent data. If more than one transmission of urgent data is received from the peer, they are all put into the normal stream. This is intended for applications that cannot afford to miss out on any urgent data but are usually too slow to respond to them promptly.

**RETURNS** OK, or **ERROR** if there is an invalid socket, an unknown option, an option length greater than `MLEN`, insufficient `mbufs`, or the call is unable to set the specified option.

**SEE ALSO** `sockLib`

---

## setvbuf()

**NAME** `setvbuf()` – specify buffering for a stream (ANSI)

**SYNOPSIS**

```
int setvbuf
(
    FILE * fp,                /* stream to set buffering for */
    char * buf,              /* buffer to use (optional) */
    int mode,                /* _IOFBF = fully buffered _IOLBF = line */
                                /* buffered _IONBF = unbuffered */
    size_t size              /* buffer size */
)
```

**DESCRIPTION** This routine sets the buffer size and buffering mode for a specified stream. It should be called only after the stream has been associated with an open file and before any other operation is performed on the stream. The argument *mode* determines how the stream will be buffered, as follows:

**\_IOFBF**  
input/output is to be fully buffered.

**\_IOLBF**  
input/output is to be line buffered.

**\_IONBF**  
input/output is to be unbuffered.

If *buf* is not a null pointer, the array it points to may be used instead of a buffer allocated by `setvbuf()`. The argument *size* specifies the size of the array. The contents of the array at any time are indeterminate.

**INCLUDE FILES** `stdio.h`

**RETURNS** Zero, or non-zero if *mode* is invalid or the request cannot be honored.

**SEE ALSO** `ansiStdio`

**shell()**

---

**shell()****NAME** shell() – the shell entry point

**SYNOPSIS**

```
void shell
(
    BOOL interactive          /* should be TRUE, except for a script */
)
```

**DESCRIPTION** This routine is the shell task. It is started with a single parameter that indicates whether this is an interactive shell to be used from a terminal or a socket, or a shell that executes a script.

Normally, the shell is spawned in interactive mode by the root task, **usrRoot()**, when VxWorks starts up. After that, **shell()** is called only to execute scripts, or when the shell is restarted after an abort.

The shell gets its input from standard input and sends output to standard output. Both standard input and standard output are initially assigned to the console, but are redirected by **telnetdTask()** and **rlogindTask()**.

The shell is not reentrant, since **yacc** does not generate a reentrant parser. Therefore, there can be only a single shell executing at one time.

**RETURNS** N/A**SEE ALSO** **shellLib**, *VxWorks Programmer's Guide: Target Shell*

---

**shellHistory()****NAME** shellHistory() – display or set the size of shell history

**SYNOPSIS**

```
void shellHistory
(
    int size                  /* 0 = display, >0 = set history to new size */
)
```

**DESCRIPTION** This routine displays shell history, or resets the default number of commands displayed by shell history to *size*. By default, history size is 20 commands. Shell history is actually maintained by **ledLib**.

**RETURNS** N/A

**SEE ALSO** **shellLib**, **ledLib**, **h()**, *VxWorks Programmer's Guide: Target Shell*, **windsh**, *Tornado User's Guide: Shell*

---

## shellInit()

**NAME** **shellInit()** – start the shell

**SYNOPSIS**

```
STATUS shellInit
(int stackSize,          /* shell stack (0 = previous/default value) */
int arg                /* argument to shell task */
)
```

**DESCRIPTION** This routine starts the shell task. If the configuration macro **INCLUDE\_SHELL** is defined, **shellInit()** is called by the root task, **usrRoot()**, in **usrConfig.c**.

**RETURNS** OK or ERROR.

**SEE ALSO** **shellLib**, *VxWorks Programmer's Guide: Target Shell*

---

## shellLock()

**NAME** **shellLock()** – lock access to the shell

**SYNOPSIS**

```
BOOL shellLock
(BOOL request          /* TRUE = lock, FALSE = unlock */
)
```

**DESCRIPTION** This routine locks or unlocks access to the shell. When locked, cooperating tasks, such as **telnetdTask()** and **rlogindTask()**, will not take the shell.

**RETURNS** TRUE if *request* is "lock" and the routine successfully locks the shell, otherwise FALSE.  
TRUE if *request* is "unlock" and the routine successfully unlocks the shell, otherwise FALSE.

**SEE ALSO** **shellLib**, *VxWorks Programmer's Guide: Target Shell*

## shellOrigStdSet()

**NAME** shellOrigStdSet() – set the shell’s default input/output/error file descriptors

**SYNOPSIS**

```
void shellOrigStdSet
(
    int which,          /* STD_IN, STD_OUT, STD_ERR */
    int fd             /* fd to be default */
)
```

**DESCRIPTION** This routine is called to change the shell’s default standard input/output/error file descriptor. Normally, it is used only by the shell, **rlogindTask()**, and **telnetdTask()**. Values for *which* can be **STD\_IN**, **STD\_OUT**, or **STD\_ERR**, as defined in **vxWorks.h**. Values for *fd* can be the file descriptor for any file or device.

**RETURNS** N/A

**SEE ALSO** shellLib

---

## shellPromptSet()

**NAME** shellPromptSet() – change the shell prompt

**SYNOPSIS**

```
void shellPromptSet
(
    char * newPrompt   /* string to become new shell prompt */
)
```

**DESCRIPTION** This routine changes the shell prompt string to *newPrompt*.

**RETURNS** N/A

**SEE ALSO** shellLib, *VxWorks Programmer’s Guide: Target Shell*, **windsh**, *Tornado User’s Guide: Shell*

---

## shellScriptAbort()

|                    |                                                                                                                                  |
|--------------------|----------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>        | <b>shellScriptAbort()</b> – signal the shell to stop processing a script                                                         |
| <b>SYNOPSIS</b>    | <b>void shellScriptAbort (void)</b>                                                                                              |
| <b>DESCRIPTION</b> | This routine signals the shell to abort processing a script file. It can be called from within a script if an error is detected. |
| <b>RETURNS</b>     | N/A                                                                                                                              |
| <b>SEE ALSO</b>    | <b>shellLib</b> , <i>VxWorks Programmer's Guide: Target Shell</i>                                                                |

---

## show()

|                    |                                                                                                                                                                                                                                                                                                                                                                                   |
|--------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>        | <b>show()</b> – print information on a specified object                                                                                                                                                                                                                                                                                                                           |
| <b>SYNOPSIS</b>    | <pre>void show (     int objId,           /* object ID */     int level           /* information level */ )</pre>                                                                                                                                                                                                                                                                 |
| <b>DESCRIPTION</b> | This command prints information on the specified object. System objects include tasks, local and shared semaphores, local and shared message queues, local and shared memory partitions, watchdogs, and symbol tables. An information level is interpreted by the objects show routine on a class by class basis. Refer to the object's library manual page for more information. |
| <b>RETURNS</b>     | N/A                                                                                                                                                                                                                                                                                                                                                                               |
| <b>SEE ALSO</b>    | <b>usrLib</b> , <b>i()</b> , <b>ti()</b> , <b>lkup()</b> , <i>VxWorks Programmer's Guide: Target Shell</i> , <b>windsh</b> , <i>Tornado User's Guide: Shell</i>                                                                                                                                                                                                                   |

---

## shutdown()

|                    |                                                                                                                                                                                                                                                                           |
|--------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>        | <b>shutdown()</b> – shut down a network connection                                                                                                                                                                                                                        |
| <b>SYNOPSIS</b>    | <pre><b>STATUS</b> shutdown (     int s,                /* socket to shut down */     int how              /* 0 = receives disallowed */                         /* 1 = sends disallowed */                         /* 2 = sends and receives disallowed */ )</pre>       |
| <b>DESCRIPTION</b> | This routine shuts down all, or part, of a connection-based socket <i>s</i> . If the value of <i>how</i> is 0, receives are disallowed. If <i>how</i> is 1, sends are disallowed. If <i>how</i> is 2, both sends and receives are disallowed.                             |
| <b>RETURNS</b>     | <b>ERROR</b> if the socket is invalid or has no registered socket-specific routines; otherwise <b>shutdown()</b> returns the return value from the socket-specific shutdown routine (typically <b>OK</b> in the case of a successful shutdown or <b>ERROR</b> otherwise). |
| <b>SEE ALSO</b>    | <b>sockLib</b>                                                                                                                                                                                                                                                            |

---

## sigaction()

|                    |                                                                                                                                                                                                                                   |
|--------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>        | <b>sigaction()</b> – examine and/or specify the action associated with a signal (POSIX)                                                                                                                                           |
| <b>SYNOPSIS</b>    | <pre>int sigaction (     int                signo, /* signal of handler of interest */     const struct sigaction * pAct, /* location of new handler */     struct sigaction *  pOact /* location to store old handler */ )</pre> |
| <b>DESCRIPTION</b> | This routine allows the calling process to examine and/or specify the action to be associated with a specific signal.                                                                                                             |
| <b>RETURNS</b>     | <b>OK</b> (0), or <b>ERROR</b> (-1) if the signal number is invalid.                                                                                                                                                              |
| <b>ERRNO</b>       | <b>EINVAL</b>                                                                                                                                                                                                                     |
| <b>SEE ALSO</b>    | <b>sigLib</b>                                                                                                                                                                                                                     |

---

## sigaddset()

|                    |                                                                                                                                               |
|--------------------|-----------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>        | <b>sigaddset()</b> – add a signal to a signal set (POSIX)                                                                                     |
| <b>SYNOPSIS</b>    | <pre>int sigaddset (     sigset_t * pSet,          /* signal set to add signal to */     int      signo           /* signal to add */ )</pre> |
| <b>DESCRIPTION</b> | This routine adds the signal specified by <i>signo</i> to the signal set specified by <i>pSet</i> .                                           |
| <b>RETURNS</b>     | OK (0), or ERROR (-1) if the signal number is invalid.                                                                                        |
| <b>ERRNO</b>       | EINVAL                                                                                                                                        |
| <b>SEE ALSO</b>    | sigLib                                                                                                                                        |

---

## sigblock()

|                    |                                                                                                                                                                                                                                                              |
|--------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>        | <b>sigblock()</b> – add to a set of blocked signals                                                                                                                                                                                                          |
| <b>SYNOPSIS</b>    | <pre>int sigblock (     int mask                 /* mask of additional signals to be blocked */ )</pre>                                                                                                                                                      |
| <b>DESCRIPTION</b> | This routine adds the signals in <i>mask</i> to the task's set of blocked signals. A one (1) in the bit mask indicates that the specified signal is blocked from delivery. Use the macro <b>SIGMASK</b> to construct the mask for a specified signal number. |
| <b>RETURNS</b>     | The previous value of the signal mask.                                                                                                                                                                                                                       |
| <b>SEE ALSO</b>    | sigLib, sigprocmask()                                                                                                                                                                                                                                        |

## sigdelset()

|                    |                                                                                                                                                       |
|--------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>        | sigdelset() – delete a signal from a signal set (POSIX)                                                                                               |
| <b>SYNOPSIS</b>    | <pre>int sigdelset (     sigset_t * pSet,          /* signal set to delete signal from */     int      signo           /* signal to delete */ )</pre> |
| <b>DESCRIPTION</b> | This routine deletes the signal specified by <i>signo</i> from the signal set specified by <i>pSet</i> .                                              |
| <b>RETURNS</b>     | OK (0), or ERROR (-1) if the signal number is invalid.                                                                                                |
| <b>ERRNO</b>       | EINVAL                                                                                                                                                |
| <b>SEE ALSO</b>    | sigLib                                                                                                                                                |

---

## sigemptyset()

|                    |                                                                                                        |
|--------------------|--------------------------------------------------------------------------------------------------------|
| <b>NAME</b>        | sigemptyset() – initialize a signal set with no signals included (POSIX)                               |
| <b>SYNOPSIS</b>    | <pre>int sigemptyset (     sigset_t * pSet          /* signal set to initialize */ )</pre>             |
| <b>DESCRIPTION</b> | This routine initializes the signal set specified by <i>pSet</i> , such that all signals are excluded. |
| <b>RETURNS</b>     | OK (0), or ERROR (-1) if the signal set cannot be initialized.                                         |
| <b>ERRNO</b>       | No errors are detectable.                                                                              |
| <b>SEE ALSO</b>    | sigLib                                                                                                 |

---

## sigfillset()

|                    |                                                                                                        |
|--------------------|--------------------------------------------------------------------------------------------------------|
| <b>NAME</b>        | <b>sigfillset()</b> – initialize a signal set with all signals included (POSIX)                        |
| <b>SYNOPSIS</b>    | <pre>int sigfillset (     sigset_t * pSet          /* signal set to initialize */ )</pre>              |
| <b>DESCRIPTION</b> | This routine initializes the signal set specified by <i>pSet</i> , such that all signals are included. |
| <b>RETURNS</b>     | OK (0), or <b>ERROR</b> (-1) if the signal set cannot be initialized.                                  |
| <b>ERRNO</b>       | No errors are detectable.                                                                              |
| <b>SEE ALSO</b>    | <b>sigLib</b>                                                                                          |

---

## sigInit()

|                    |                                                                                                                                                                             |
|--------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>        | <b>sigInit()</b> – initialize the signal facilities                                                                                                                         |
| <b>SYNOPSIS</b>    | <pre>int sigInit (void)</pre>                                                                                                                                               |
| <b>DESCRIPTION</b> | This routine initializes the signal facilities. It is usually called from the system start-up routine <b>usrInit()</b> in <i>usrConfig</i> , before interrupts are enabled. |
| <b>RETURNS</b>     | OK, or <b>ERROR</b> if the delete hooks cannot be installed.                                                                                                                |
| <b>ERRNO</b>       | <b>S_taskLib_TASK_HOOK_TABLE_FULL</b>                                                                                                                                       |
| <b>SEE ALSO</b>    | <b>sigLib</b>                                                                                                                                                               |

## sigismember()

|                    |                                                                                                                                             |
|--------------------|---------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>        | <b>sigismember()</b> – test to see if a signal is in a signal set (POSIX)                                                                   |
| <b>SYNOPSIS</b>    | <pre>int sigismember (     const sigset_t * pSet,    /* signal set to test */     int             signo    /* signal to test for */ )</pre> |
| <b>DESCRIPTION</b> | This routine tests whether the signal specified by <i>signo</i> is a member of the set specified by <i>pSet</i> .                           |
| <b>RETURNS</b>     | 1 if the specified signal is a member of the specified set, <b>OK</b> (0) if it is not, or <b>ERROR</b> (-1) if the test fails.             |
| <b>ERRNO</b>       | <b>EINVAL</b>                                                                                                                               |
| <b>SEE ALSO</b>    | <b>sigLib</b>                                                                                                                               |

---

## signal()

|                    |                                                                                                                                                                                                                                                                                                                                                                                                     |
|--------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>        | <b>signal()</b> – specify the handler associated with a signal                                                                                                                                                                                                                                                                                                                                      |
| <b>SYNOPSIS</b>    | <pre>void (*signal (     int signo,     void (*pHandler) () )) ()</pre>                                                                                                                                                                                                                                                                                                                             |
| <b>DESCRIPTION</b> | This routine chooses one of three ways in which receipt of the signal number <i>signo</i> is to be subsequently handled. If the value of <i>pHandler</i> is <b>SIG_DFL</b> , default handling for that signal will occur. If the value of <i>pHandler</i> is <b>SIG_IGN</b> , the signal will be ignored. Otherwise, <i>pHandler</i> must point to a function to be called when that signal occurs. |
| <b>RETURNS</b>     | The value of the previous signal handler, or <b>SIG_ERR</b> .                                                                                                                                                                                                                                                                                                                                       |
| <b>SEE ALSO</b>    | <b>sigLib</b>                                                                                                                                                                                                                                                                                                                                                                                       |

---

## sigpending()

|                    |                                                                                                                                                             |
|--------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>        | <b>sigpending()</b> – retrieve the set of pending signals blocked from delivery (POSIX)                                                                     |
| <b>SYNOPSIS</b>    | <pre>int sigpending (     sigset_t * pSet          /* location to store pending signal set */ )</pre>                                                       |
| <b>DESCRIPTION</b> | This routine stores the set of signals that are blocked from delivery and that are pending for the calling process in the space pointed to by <i>pSet</i> . |
| <b>RETURNS</b>     | OK (0), or ERROR (-1) if the signal TCB cannot be allocated.                                                                                                |
| <b>ERRNO</b>       | ENOMEM                                                                                                                                                      |
| <b>SEE ALSO</b>    | sigLib                                                                                                                                                      |

---

## sigprocmask()

|                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
|--------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>        | <b>sigprocmask()</b> – examine and/or change the signal mask (POSIX)                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| <b>SYNOPSIS</b>    | <pre>int sigprocmask (     int          how,          /* how signal mask will be changed */     const sigset_t * pSet,     /* location of new signal mask */     sigset_t *   pOset        /* location to store old signal mask */ )</pre>                                                                                                                                                                                                                                                                                                                                                                                                           |
| <b>DESCRIPTION</b> | <p>This routine allows the calling process to examine and/or change its signal mask. If the value of <i>pSet</i> is not NULL, it points to a set of signals to be used to change the currently blocked set.</p> <p>The value of <i>how</i> indicates the manner in which the set is changed and consists of one of the following, defined in <b>signal.h</b>:</p> <p><b>SIG_BLOCK</b><br/>the resulting set is the union of the current set and the signal set pointed to by <i>pSet</i>.</p> <p><b>SIG_UNBLOCK</b><br/>the resulting set is the intersection of the current set and the complement of the signal set pointed to by <i>pSet</i>.</p> |

**SIG\_SETMASK**

the resulting set is the signal set pointed to by *pSset*.

**RETURNS** OK (0), or **ERROR** (-1) if *how* is invalid.

**ERRNO** EINVAL

**SEE ALSO** sigLib, sigsetmask(), sigblock()

---

## sigqueue()

**NAME** sigqueue() – send a queued signal to a task

**SYNOPSIS**

```
int sigqueue
(
    int          tid,
    int          signo,
    const union sigval value
)
```

**DESCRIPTION** The function **sigqueue()** sends the signal specified by *signo* with the signal-parameter value specified by *value* to the process specified by *tid*.

**RETURNS** OK (0), or **ERROR** (-1) if the task ID or signal number is invalid, or if there are no queued-signal buffers available.

**ERRNO** EINVAL, EAGAIN

**SEE ALSO** sigLib

---

## sigqueueInit()

**NAME** `sigqueueInit()` – initialize the queued signal facilities

**SYNOPSIS**

```
int sigqueueInit
(
    int nQueues
)
```

**DESCRIPTION** This routine initializes the queued signal facilities. It must be called before any call to `sigqueue()`. It is usually called from the system start-up routine `usrInit()` in `usrConfig`, after `sysInit()` is called.

It allocates `nQueues` buffers to be used by `sigqueue()`. A buffer is used by each call to `sigqueue()` and freed when the signal is delivered (thus if a signal is block, the buffer is unavailable until the signal is unblocked.)

**RETURNS** OK, or `ERROR` if memory could not be allocated.

**SEE ALSO** `sigLib`

---

## sigsetmask()

**NAME** `sigsetmask()` – set the signal mask

**SYNOPSIS**

```
int sigsetmask
(
    int mask                /* new signal mask */
)
```

**DESCRIPTION** This routine sets the calling task's signal mask to a specified value. A one (1) in the bit mask indicates that the specified signal is blocked from delivery. Use the macro `SIGMASK` to construct the mask for a specified signal number.

**RETURNS** The previous value of the signal mask.

**SEE ALSO** `sigLib`, `sigprocmask()`

## sigsuspend()

**NAME** sigsuspend() – suspend the task until delivery of a signal (POSIX)

**SYNOPSIS**

```
int sigsuspend
(
    const sigset_t * pSet    /* signal mask while suspended */
)
```

**DESCRIPTION** This routine suspends the task until delivery of a signal. While suspended, *pSet* is used as the set of masked signals.

---

**NOTE:** Since the **sigsuspend()** function suspends thread execution indefinitely, there is no successful completion return value.

---

**RETURNS** -1, always.

**ERRNO** EINTR

**SEE ALSO** sigLib

---

## sigtimedwait()

**NAME** sigtimedwait() – wait for a signal

**SYNOPSIS**

```
int sigtimedwait
(
    const sigset_t *    pSet,    /* the signal mask while suspended */
    struct siginfo *    pInfo,   /* return value */
    const struct timespec * pTimeout
)
```

**DESCRIPTION** The function **sigtimedwait()** selects the pending signal from the set specified by *pSet*. If multiple signals in *pSet* are pending, it will remove and return the lowest numbered one. If no signal in *pSet* is pending at the time of the call, the task will be suspend until one of the signals in *pSet* become pending, it is interrupted by an unblocked caught signal, or until the time interval specified by *pTimeout* has expired. If *pTimeout* is **NULL**, then the timeout interval is forever.

If the *pInfo* argument is non-NULL, the selected signal number is stored in the **si\_signo** member, and the cause of the signal is stored in the **si\_code** member. If the signal is a queued signal, the value is stored in the **si\_value** member of *pInfo*; otherwise the content of **si\_value** is undefined.

The following values are defined in **signal.h** for **si\_code**:

**SI\_USER**

the signal was sent by the **kill()** function.

**SI\_QUEUE**

the signal was sent by the **sigqueue()** function.

**SI\_TIMER**

the signal was generated by the expiration of a timer set by **timer\_settime()**.

**SI\_ASYNCIO**

the signal was generated by the completion of an asynchronous I/O request.

**SI\_MESGQ**

the signal was generated by the arrival of a message on an empty message queue.

The function **sigtimedwait()** provides a synchronous mechanism for tasks to wait for asynchronously generated signals. A task should use **sigprocmask()** to block any signals it wants to handle synchronously and leave their signal handlers in the default state. The task can then make repeated calls to **sigtimedwait()** to remove any signals that are sent to it.

**RETURNS**

Upon successful completion (that is, one of the signals specified by *pSet* is pending or is generated) **sigtimedwait()** will return the selected signal number. Otherwise, a value of -1 is returned and **errno** is set to indicate the error.

**ERRNO**

**EINTR**

The wait was interrupted by an unblocked, caught signal.

**EAGAIN**

No signal specified by *pSet* was delivered within the specified timeout period.

**EINVAL**

The *pTimeout* argument specified a **tv\_nsec** value less than zero or greater than or equal to 1000 million.

**SEE ALSO**

**sigLib**, **sigwait()**

## sigvec()

**NAME** sigvec() – install a signal handler

**SYNOPSIS**

```
int sigvec
(
    int          sig, /* signal to attach handler to */
    const struct sigvec * pVec, /* new handler information */
    struct sigvec *   pOvec /* previous handler information */
)
```

**DESCRIPTION** This routine binds a signal handler routine referenced by *pVec* to a specified signal *sig*. It can also be used to determine which handler, if any, has been bound to a particular signal: **sigvec()** copies current signal handler information for *sig* to *pOvec* and does not install a signal handler if *pVec* is set to **NULL** (0).

Both *pVec* and *pOvec* are pointers to a structure of type **struct sigvec**. The information passed includes not only the signal handler routine, but also the signal mask and additional option bits. The structure **sigvec** and the available options are defined in **signal.h**.

**RETURNS** OK (0), or **ERROR** (-1) if the signal number is invalid or the signal TCB cannot be allocated.

**ERRNO** EINVAL, ENOMEM

**SEE ALSO** sigLib

---

## sigwait()

**NAME** sigwait() – wait for a signal to be delivered (POSIX)

**SYNOPSIS**

```
int sigwait
(
    const sigset_t * pSet,
    int *          pSig
)
```

**DESCRIPTION** This routine waits until one of the signals specified in *pSet* is delivered to the calling thread. It then stores the number of the signal received in the location pointed to by *pSig*.

The signals in *pSet* must not be ignored on entrance to **sigwait()**. If the delivered signal has a signal handler function attached, that function is not called.

**RETURNS** OK, or **ERROR** on failure.

**SEE ALSO** sigLib, sigtimedwait()

---

## sigwaitinfo()

**NAME** sigwaitinfo() – wait for real-time signals

**SYNOPSIS**

```
int sigwaitinfo
(
    const sigset_t * pSet,      /* the signal mask while suspended */
    struct siginfo * pInfo     /* return value */
)
```

**DESCRIPTION** The function **sigwaitinfo()** is equivalent to calling **sigtimedwait()** with *pTimeout* equal to **NULL**. See that manual entry for more information.

**RETURNS** Upon successful completion (that is, one of the signals specified by *pSet* is pending or is generated) **sigwaitinfo()** returns the selected signal number. Otherwise, a value of -1 is returned and **errno** is set to indicate the error.

**ERRNO** **EINTR**  
The wait was interrupted by an unblocked, caught signal.

**SEE ALSO** sigLib

## **sin()**

|                      |                                                                                                   |
|----------------------|---------------------------------------------------------------------------------------------------|
| <b>NAME</b>          | <b>sin()</b> – compute a sine (ANSI)                                                              |
| <b>SYNOPSIS</b>      | <pre>double sin (     double x                /* angle in radians */ )</pre>                      |
| <b>DESCRIPTION</b>   | This routine computes the sine of $x$ in double precision. The angle $x$ is expressed in radians. |
| <b>INCLUDE FILES</b> | <b>math.h</b>                                                                                     |
| <b>RETURNS</b>       | The double-precision sine of $x$ .                                                                |
| <b>SEE ALSO</b>      | <b>ansiMath, mathALib</b>                                                                         |

---

## **sincos()**

|                      |                                                                                                                                                                                 |
|----------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>          | <b>sincos()</b> – compute both a sine and cosine                                                                                                                                |
| <b>SYNOPSIS</b>      | <pre>void sincos (     double x,                /* angle in radians */     double *sinResult, /* sine result buffer */     double *cosResult /* cosine result buffer */ )</pre> |
| <b>DESCRIPTION</b>   | This routine computes both the sine and cosine of $x$ in double precision. The sine is copied to <i>sinResult</i> and the cosine is copied to <i>cosResult</i> .                |
| <b>INCLUDE FILES</b> | <b>math.h</b>                                                                                                                                                                   |
| <b>RETURNS</b>       | N/A                                                                                                                                                                             |
| <b>SEE ALSO</b>      | <b>mathALib</b>                                                                                                                                                                 |

---

## **sincosf()**

|                      |                                                                                                                                                                                                         |
|----------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>          | <b>sincosf()</b> – compute both a sine and cosine                                                                                                                                                       |
| <b>SYNOPSIS</b>      | <pre>void sincosf (     float x,          /* angle in radians */     float *sinResult, /* sine result buffer */     float *cosResult /* cosine result buffer */ )</pre>                                 |
| <b>DESCRIPTION</b>   | This routine computes both the sine and cosine of $x$ in single precision. The sine is copied to <i>sinResult</i> and the cosine is copied to <i>cosResult</i> . The angle $x$ is expressed in radians. |
| <b>INCLUDE FILES</b> | <b>math.h</b>                                                                                                                                                                                           |
| <b>RETURNS</b>       | N/A                                                                                                                                                                                                     |
| <b>SEE ALSO</b>      | <b>mathALib</b>                                                                                                                                                                                         |

---

## **sinf()**

|                      |                                                                                                  |
|----------------------|--------------------------------------------------------------------------------------------------|
| <b>NAME</b>          | <b>sinf()</b> – compute a sine (ANSI)                                                            |
| <b>SYNOPSIS</b>      | <pre>float sinf (     float x /* angle in radians */ )</pre>                                     |
| <b>DESCRIPTION</b>   | This routine returns the sine of $x$ in single precision. The angle $x$ is expressed in radians. |
| <b>INCLUDE FILES</b> | <b>math.h</b>                                                                                    |
| <b>RETURNS</b>       | The single-precision sine of $x$ .                                                               |
| <b>SEE ALSO</b>      | <b>mathALib</b>                                                                                  |

**sinh()**

---

**sinh()**

|                      |                                                                                                                                       |
|----------------------|---------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>          | <b>sinh()</b> – compute a hyperbolic sine (ANSI)                                                                                      |
| <b>SYNOPSIS</b>      | <pre>double sinh (     double x          /* number whose hyperbolic sine is required */ )</pre>                                       |
| <b>DESCRIPTION</b>   | This routine returns the hyperbolic sine of $x$ in double precision (IEEE double, 53 bits). A range error occurs if $x$ is too large. |
| <b>INCLUDE FILES</b> | <b>math.h</b>                                                                                                                         |
| <b>RETURNS</b>       | The double-precision hyperbolic sine of $x$ .<br>Special cases:<br>If $x$ is +INF, -INF, or NaN, <b>sinh()</b> returns $x$ .          |
| <b>SEE ALSO</b>      | <b>ansiMath, mathALib</b>                                                                                                             |

---

**sinhf()**

|                      |                                                                                                |
|----------------------|------------------------------------------------------------------------------------------------|
| <b>NAME</b>          | <b>sinhf()</b> – compute a hyperbolic sine (ANSI)                                              |
| <b>SYNOPSIS</b>      | <pre>float sinhf (     float x          /* number whose hyperbolic sine is required */ )</pre> |
| <b>DESCRIPTION</b>   | This routine returns the hyperbolic sine of $x$ in single precision.                           |
| <b>INCLUDE FILES</b> | <b>math.h</b>                                                                                  |
| <b>RETURNS</b>       | The single-precision hyperbolic sine of $x$ .                                                  |
| <b>SEE ALSO</b>      | <b>mathALib</b>                                                                                |

---

## sleep()

|                    |                                                                                                                                                                                                                                                                                                             |
|--------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>        | <b>sleep()</b> – delay for a specified amount of time                                                                                                                                                                                                                                                       |
| <b>SYNOPSIS</b>    | <pre>unsigned int sleep (     unsigned int secs )</pre>                                                                                                                                                                                                                                                     |
| <b>DESCRIPTION</b> | <p>This routine causes the calling task to be blocked for <i>secs</i> seconds.</p> <p>The time the task is blocked for may be longer than requested due to the rounding up of the request to the timer's resolution or to other scheduling activities (<i>e.g.</i>, a higher priority task intervenes).</p> |
| <b>RETURNS</b>     | Zero if the requested time has elapsed, or the number of seconds remaining if it was interrupted.                                                                                                                                                                                                           |
| <b>ERRNO</b>       | EINVAL, EINTR                                                                                                                                                                                                                                                                                               |
| <b>SEE ALSO</b>    | <b>timerLib</b> , <b>nanosleep()</b> , <b>taskDelay()</b>                                                                                                                                                                                                                                                   |

---

## smMemAddToPool()

|                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
|--------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>        | <b>smMemAddToPool()</b> – add memory to shared memory system partition (VxMP Opt.)                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| <b>SYNOPSIS</b>    | <pre>STATUS smMemAddToPool (     char * pPool,          /* pointer to memory pool */     unsigned poolSize     /* block size in bytes */ )</pre>                                                                                                                                                                                                                                                                                                                                                                                          |
| <b>DESCRIPTION</b> | <p>This routine adds memory to the shared memory system partition after the initial allocation of memory. The memory added need not be contiguous with memory previously assigned, but it must be in the same address space.</p> <p><i>pPool</i> is the global address of shared memory added to the partition. The memory area pointed to by <i>pPool</i> must be in the same address space as the shared memory anchor and shared memory pool.</p> <p><i>poolSize</i> is the size in bytes of shared memory added to the partition.</p> |

|                     |                                                                                                         |
|---------------------|---------------------------------------------------------------------------------------------------------|
| <b>AVAILABILITY</b> | This routine is distributed as a component of the unbundled shared memory objects support option, VxMP. |
| <b>RETURNS</b>      | OK, or ERROR if access to the shared memory system partition fails.                                     |
| <b>ERRNO</b>        | S_smObjLib_LOCK_TIMEOUT                                                                                 |
| <b>SEE ALSO</b>     | smMemLib                                                                                                |

---

## smMemCalloc()

|                     |                                                                                                                                                                                                                                          |
|---------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>         | <b>smMemCalloc()</b> – allocate memory for array from shared memory system partition (VxMP Opt.)                                                                                                                                         |
| <b>SYNOPSIS</b>     | <pre>void * smMemCalloc (     int elemNum,          /* number of elements */     int elemSize         /* size of elements */ )</pre>                                                                                                     |
| <b>DESCRIPTION</b>  | This routine allocates a block of memory for an array that contains <i>elemNum</i> elements of size <i>elemSize</i> from the shared memory system partition. The return value is the local address of the allocated shared memory block. |
| <b>AVAILABILITY</b> | This routine is distributed as a component of the unbundled shared memory objects support option, VxMP.                                                                                                                                  |
| <b>RETURNS</b>      | A pointer to the block, or NULL if the memory cannot be allocated.                                                                                                                                                                       |
| <b>ERRNO</b>        | S_memLib_NOT_ENOUGH_MEMORY<br>S_smObjLib_LOCK_TIMEOUT                                                                                                                                                                                    |
| <b>SEE ALSO</b>     | smMemLib                                                                                                                                                                                                                                 |

---

## smMemFindMax()

|                     |                                                                                                                   |
|---------------------|-------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>         | <b>smMemFindMax()</b> – find largest free block in shared memory system partition (VxMP)                          |
| <b>SYNOPSIS</b>     | <pre>int smMemFindMax (void)</pre>                                                                                |
| <b>DESCRIPTION</b>  | This routine searches for the largest block in the shared memory system partition free list and returns its size. |
| <b>AVAILABILITY</b> | This routine is distributed as a component of the unbundled shared memory objects support option, VxMP.           |
| <b>RETURNS</b>      | The size (in bytes) of the largest available block, or <b>ERROR</b> if the attempt to access the partition fails. |
| <b>ERRNO</b>        | <b>S_smObjLib_LOCK_TIMEOUT</b>                                                                                    |
| <b>SEE ALSO</b>     | <b>smMemLib</b>                                                                                                   |

---

## smMemFree()

|                     |                                                                                                                                                                                                                                                      |
|---------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>         | <b>smMemFree()</b> – free a shared memory system partition block of memory (VxMP Opt.)                                                                                                                                                               |
| <b>SYNOPSIS</b>     | <pre>STATUS smMemFree<br/>(<br/>    void * ptr                /* pointer to block of memory to be freed */<br/>)</pre>                                                                                                                               |
| <b>DESCRIPTION</b>  | This routine takes a block of memory previously allocated with <b>smMemMalloc()</b> or <b>smMemCalloc()</b> and returns it to the free shared memory system pool.<br><br>It is an error to free a block of memory that was not previously allocated. |
| <b>AVAILABILITY</b> | This routine is distributed as a component of the unbundled shared memory objects support option, VxMP.                                                                                                                                              |
| <b>RETURNS</b>      | OK, or <b>ERROR</b> if the block is invalid.                                                                                                                                                                                                         |
| <b>ERRNO</b>        | <b>S_memLib_BLOCK_ERROR</b> , <b>S_smObjLib_LOCK_TIMEOUT</b>                                                                                                                                                                                         |
| <b>SEE ALSO</b>     | <b>smMemLib</b> , <b>smMemMalloc()</b> , <b>smMemCalloc()</b>                                                                                                                                                                                        |

---

## smMemMalloc()

|                     |                                                                                                                                                                                                                     |
|---------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>         | <b>smMemMalloc()</b> – allocate block of memory from shared memory system partition (VxMP Opt.)                                                                                                                     |
| <b>SYNOPSIS</b>     | <pre>void * smMemMalloc (     unsigned nBytes          /* number of bytes to allocate */ )</pre>                                                                                                                    |
| <b>DESCRIPTION</b>  | This routine allocates a block of memory from the shared memory system partition whose size is equal to or greater than <i>nBytes</i> . The return value is the local address of the allocated shared memory block. |
| <b>AVAILABILITY</b> | This routine is distributed as a component of the unbundled shared memory objects support option, VxMP.                                                                                                             |
| <b>RETURNS</b>      | A pointer to the block, or NULL if the memory cannot be allocated.                                                                                                                                                  |
| <b>ERRNO</b>        | <b>S_memLib_NOT_ENOUGH_MEMORY</b><br><b>S_smObjLib_LOCK_TIMEOUT</b>                                                                                                                                                 |
| <b>SEE ALSO</b>     | <b>smMemLib</b>                                                                                                                                                                                                     |

---

## smMemOptionsSet()

|                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
|--------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>        | <b>smMemOptionsSet()</b> – set debug options for shared memory system partition (VxMP Opt.)                                                                                                                                                                                                                                                                                                                                                                        |
| <b>SYNOPSIS</b>    | <pre>STATUS smMemOptionsSet (     unsigned options          /* options for system partition */ )</pre>                                                                                                                                                                                                                                                                                                                                                             |
| <b>DESCRIPTION</b> | This routine sets the debug options for the shared system memory partition. Two kinds of errors are detected: attempts to allocate more memory than is available, and bad blocks found when memory is freed or reallocated. In both cases, the following options can be selected for actions to be taken when an error is detected: (1) return the error status, (2) log an error message and return the error status, or (3) log an error message and suspend the |

calling task. These options are discussed in detail in the library manual entry for **smMemLib**.

|                     |                                                                                                         |
|---------------------|---------------------------------------------------------------------------------------------------------|
| <b>AVAILABILITY</b> | This routine is distributed as a component of the unbundled shared memory objects support option, VxMP. |
| <b>RETURNS</b>      | OK or ERROR.                                                                                            |
| <b>ERRNO</b>        | S_smObjLib_LOCK_TIMEOUT                                                                                 |
| <b>SEE ALSO</b>     | smMemLib                                                                                                |

---

## smMemRealloc()

**NAME** **smMemRealloc()** – reallocate block of memory from shared memory system partition (VxMP Opt.)

**SYNOPSIS**

```
void * smMemRealloc
(
    void * pBlock,          /* block to be reallocated */
    unsigned newSize       /* new block size */
)
```

**DESCRIPTION** This routine changes the size of a specified block and returns a pointer to the new block of shared memory. The contents that fit inside the new size (or old size, if smaller) remain unchanged. The return value is the local address of the reallocated shared memory block.

**AVAILABILITY** This routine is distributed as a component of the unbundled shared memory objects support option, VxMP.

**RETURNS** A pointer to the new block of memory, or NULL if the reallocation cannot be completed.

**ERRNO** S\_memLib\_NOT\_ENOUGH\_MEMORY  
 S\_memLib\_BLOCK\_ERROR  
 S\_smObjLib\_LOCK\_TIMEOUT

**SEE ALSO** smMemLib



---

## smMemShow()

**NAME** **smMemShow()** – show the shared memory system partition blocks and statistics (VxMP Opt.)

**SYNOPSIS**

```
void smMemShow
(
    int type                /* 0 = statistics, 1 = statistics & list */
)
```

**DESCRIPTION**

This routine displays the total amount of free space in the shared memory system partition, including number of blocks, average block size, and maximum block size. It also shows the number of blocks currently allocated, and the average allocated block size.

If *type* is 1, it displays a list of all the blocks in the free list of the shared memory system partition.

---

**WARNING:** This routine locks access to the shared memory system partition while displaying the information. This can compromise the access time to the partition from other CPUs in the system. Generally, this routine is used for debugging purposes only.

---

**EXAMPLE**

```
-> smMemShow 1
FREE LIST:
  num      addr          size
  -----
   1  0x4ffef0          264
   2  0x4fef18         1700
SUMMARY:
  status      bytes      blocks  ave block  max block
  -----
  current
  free        1964         2         982      1700
  alloc       2356         1        2356         -
  cumulative
  alloc       2620         2        1310         -
value = 0 = 0x0
```

**AVAILABILITY** This routine is distributed as a component of the unbundled shared memory objects support option, VxMP.

**RETURNS** N/A

**SEE ALSO** **smMemShow**, **windsh**, *VxWorks Programmer's Guide: Target Shell*, *Tornado User's Guide: Shell*

---

## smNameAdd()

**NAME** `smNameAdd()` – add a name to the shared memory name database (VxMP Opt.)

**SYNOPSIS**

```
STATUS smNameAdd
(
    char * name,           /* name string to enter in database */
    void * value,         /* value associated with name */
    int  type             /* type associated with name */
)
```

**DESCRIPTION** This routine adds a name of specified object type and value to the shared memory objects name database.

The *name* parameter is an arbitrary null-terminated string with a maximum of 20 characters, including EOS.

By convention, *type* values of less than 0x1000 are reserved by VxWorks; all other values are user definable. The following types are predefined in **smNameLib.h**:

| Name         | Value | Type                          |
|--------------|-------|-------------------------------|
| T_SM_SEM_B   | = 0   | shared binary semaphore       |
| T_SM_SEM_C   | = 1   | shared counting semaphore     |
| T_SM_MSG_Q   | = 2   | shared message queue          |
| T_SM_PART_ID | = 3   | shared memory partition       |
| T_SM_BLOCK   | = 4   | shared memory allocated block |

A name can be entered only once in the database, but there can be more than one name associated with an object ID.

**AVAILABILITY** This routine is distributed as a component of the unbundled shared memory objects support option, VxMP.

**RETURNS** OK, or ERROR if there is insufficient memory for *name* to be allocated, if *name* is already in the database, or if the database is already full.

**ERRNO**

- S\_smNameLib\_NOT\_INITIALIZED
- S\_smNameLib\_NAME\_TOO\_LONG
- S\_smNameLib\_NAME\_ALREADY\_EXIST
- S\_smNameLib\_DATABASE\_FULL
- S\_smObjLib\_LOCK\_TIMEOUT

**SEE ALSO** `smNameLib`, `smNameShow`

---

## smNameFind()

|                     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
|---------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>         | <b>smNameFind()</b> – look up a shared memory object by name (VxMP Opt.)                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| <b>SYNOPSIS</b>     | <pre>STATUS smNameFind (     char *   name,           /* name to search for */     void * * pValue,        /* pointer where to return value */     int *    pType,         /* pointer where to return object type */     int      waitType       /* NO_WAIT or WAIT_FOREVER */ )</pre>                                                                                                                                                                                                                                                                                                                                                          |
| <b>DESCRIPTION</b>  | <p>This routine searches the shared memory objects name database for an object matching a specified <i>name</i>. If the object is found, its value and type are copied to the addresses pointed to by <i>pValue</i> and <i>pType</i>. The value of <i>waitType</i> can be one of the following:</p> <p><b>NO_WAIT (0)</b><br/>The call returns immediately, even if <i>name</i> is not in the database.</p> <p><b>WAIT_FOREVER (-1)</b><br/>The call returns only when <i>name</i> is available in the database. If <i>name</i> is not already in, the database is scanned periodically as the routine waits for <i>name</i> to be entered.</p> |
| <b>AVAILABILITY</b> | This routine is distributed as a component of the unbundled shared memory objects support option, VxMP.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| <b>RETURNS</b>      | OK, or ERROR if the object is not found, if <i>name</i> is too long, or the wait type is invalid.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
| <b>ERRNO</b>        | S_smNameLib_NOT_INITIALIZED<br>S_smNameLib_NAME_TOO_LONG<br>S_smNameLib_NAME_NOT_FOUND<br>S_smNameLib_INVALID_WAIT_TYPE<br>S_smObjLib_LOCK_TIMEOUT                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| <b>SEE ALSO</b>     | smNameLib, smNameShow                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |

---

## smNameFindByValue()

|                     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
|---------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>         | <code>smNameFindByValue()</code> – look up a shared memory object by value (VxMP Opt.)                                                                                                                                                                                                                                                                                                                                                                                                                        |
| <b>SYNOPSIS</b>     | <pre>STATUS smNameFindByValue (     void * value,           /* value to search for */     char * name,           /* pointer where to return name */     int * pType,           /* pointer where to return object type */     int waitType           /* NO_WAIT or WAIT_FOREVER */ )</pre>                                                                                                                                                                                                                     |
| <b>DESCRIPTION</b>  | <p>This routine searches the shared memory name database for an object matching a specified value. If the object is found, its name and type are copied to the addresses pointed to by <i>name</i> and <i>pType</i>. The value of <i>waitType</i> can be one of the following:</p> <p><b>NO_WAIT (0)</b><br/>The call returns immediately, even if the object value is not in the database.</p> <p><b>WAIT_FOREVER (-1)</b><br/>The call returns only when the object value is available in the database.</p> |
| <b>AVAILABILITY</b> | This routine is distributed as a component of the unbundled shared memory objects support option, VxMP.                                                                                                                                                                                                                                                                                                                                                                                                       |
| <b>RETURNS</b>      | OK, or <b>ERROR</b> if <i>value</i> is not found or if the wait type is invalid.                                                                                                                                                                                                                                                                                                                                                                                                                              |
| <b>ERRNO</b>        | <code>S_smNameLib_NOT_INITIALIZED</code><br><code>S_smNameLib_VALUE_NOT_FOUND</code><br><code>S_smNameLib_INVALID_WAIT_TYPE</code><br><code>S_smObjLib_LOCK_TIMEOUT</code>                                                                                                                                                                                                                                                                                                                                    |
| <b>SEE ALSO</b>     | <code>smNameLib</code> , <code>smNameShow</code>                                                                                                                                                                                                                                                                                                                                                                                                                                                              |

## smNameRemove()

|                     |                                                                                                                   |
|---------------------|-------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>         | <b>smNameRemove()</b> – remove an object from the shared memory objects name database (VxMP Opt.)                 |
| <b>SYNOPSIS</b>     | <pre><b>STATUS</b> smNameRemove (     char * name           /* name of object to remove */ )</pre>                |
| <b>DESCRIPTION</b>  | This routine removes an object called <i>name</i> from the shared memory objects name database.                   |
| <b>AVAILABILITY</b> | This routine is distributed as a component of the unbundled shared memory objects support option, VxMP.           |
| <b>RETURNS</b>      | OK, or ERROR if the object name is not in the database or if <i>name</i> is too long.                             |
| <b>ERRNO</b>        | S_smNameLib_NOT_INITIALIZED<br>S_smNameLib_NAME_TOO_LONG<br>S_smNameLib_NAME_NOT_FOUND<br>S_smObjLib_LOCK_TIMEOUT |
| <b>SEE ALSO</b>     | smNameLib, smNameShow                                                                                             |

---

## smNameShow()

|                    |                                                                                                                                                                                                                             |
|--------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>        | <b>smNameShow()</b> – show the contents of the shared memory objects name database (VxMP Opt.)                                                                                                                              |
| <b>SYNOPSIS</b>    | <pre><b>STATUS</b> smNameShow (     int level             /* information level */ )</pre>                                                                                                                                   |
| <b>DESCRIPTION</b> | This routine displays the names, values, and types of objects stored in the shared memory objects name database. Predefined types are shown, using their ASCII representations; all other types are printed in hexadecimal. |

The *level* parameter defines the level of database information displayed. If *level* is 0, only statistics on the database contents are displayed. If *level* is greater than 0, then both statistics and database contents are displayed.

---

**WARNING:** This routine locks access to the shared memory objects name database while displaying its contents. This can compromise the access time to the name database from other CPUs in the system. Generally, this routine is used for debugging purposes only.

---

**EXAMPLE**

```
-> smNameShow
Names in Database Max : 30 Current : 6 Free : 24
-> smNameShow 1
Names in Database Max : 30 Current : 6 Free : 24
Name                Value                Type
-----
inputImage           0x802340             SM_MEM_BLOCK
ouputImage           0x806340             SM_MEM_BLOCK
imagePool            0x802001             SM_MEM_PART
imageInSem           0x8e0001             SM_SEM_B
imageOutSem          0x8e0101             SM_SEM_C
actionQ              0x8e0201             SM_MSG_Q
userObject           0x8e0400             0x1b0
```

**AVAILABILITY**

This routine is distributed as a component of the unbundled shared memory objects support option, VxMP.

**RETURNS**

OK, or ERROR if the name facility is not initialized.

**ERRNO**

S\_smNameLib\_NOT\_INITIALIZED  
S\_smObjLib\_LOCK\_TIMEOUT

**SEE ALSO**

smNameShow, smNameLib




---

## smNetShow()

**NAME**

smNetShow() – show information about a shared memory network

**SYNOPSIS**

```
STATUS smNetShow
(
    char * ifName,          /* backplane interface name (NULL == "sm0") */
    BOOL zero               /* TRUE = zap totals */
)
```

**DESCRIPTION** This routine displays information about the different CPUs configured in a shared memory network specified by *ifName*. It prints error statistics and zeros these fields if *zero* is set to **TRUE**.

**EXAMPLE**

```
-> smNetShow
Anchor at 0x800000
heartbeat = 705, header at 0x800010, free pkts = 237.
cpu int type      arg1      arg2      arg3      queued pkts
-----
0  poll           0x0       0x0       0x0       0
1  poll           0x0       0x0       0x0       0
2  bus-int        0x3       0xc9      0x0       0
3  mbox-2         0x2d      0x8000    0x0       0
input packets = 192      output packets = 164
output errors = 0        collisions = 0
value = 1 = 0x1
```

**RETURNS** **OK**, or **ERROR** if there is a hardware setup problem or the routine cannot be initialized.

**SEE ALSO** **smNetShow**, **smNetLib**

---

## smObjAttach()

**NAME** **smObjAttach()** – attach the calling CPU to the shared memory objects facility (VxMP Opt.)

**SYNOPSIS**

```
STATUS smObjAttach
(
    SM_OBJ_DESC * pSmObjDesc /* pointer to shared memory descriptor */
)
```

**DESCRIPTION** This routine “attaches” the calling CPU to the shared memory objects facility. The shared memory area is identified by the shared memory descriptor with an address specified by *pSmObjDesc*. The descriptor must already have been initialized by calling **smObjInit()**.

This routine is called automatically when the component **INCLUDE\_SM\_OBJ** is included.

This routine will complete the attach process only if and when the shared memory has been initialized by the master CPU. If the shared memory is not recognized as active within the timeout period (10 minutes), this routine returns **ERROR**.

The **smObjAttach()** routine connects the shared memory objects handler to the shared memory interrupt. Note that this interrupt may be shared between the shared memory

network driver and the shared memory objects facility when both are used at the same time.

---

**WARNING:** Once a CPU has attached itself to the shared memory objects facility, it cannot be detached. Since the shared memory network driver and the shared memory objects facility use the same low-level attaching mechanism, a CPU cannot be detached from a shared memory network driver if the CPU also uses shared memory objects.

---

|                     |                                                                                                                    |
|---------------------|--------------------------------------------------------------------------------------------------------------------|
| <b>AVAILABILITY</b> | This routine is distributed as a component of the unbundled shared memory objects support option, VxMP.            |
| <b>RETURNS</b>      | OK, or <b>ERROR</b> if the shared memory objects facility is not active or the number of CPUs exceeds the maximum. |
| <b>ERRNO</b>        | <code>S_smLib_INVALID_CPU_NUMBER</code>                                                                            |
| <b>SEE ALSO</b>     | <code>smObjLib</code> , <code>smObjSetup()</code> , <code>smObjInit()</code>                                       |

---

## smObjGlobalToLocal()

**NAME** `smObjGlobalToLocal()` – convert a global address to a local address (VxMP Opt.)

**SYNOPSIS**

```
void * smObjGlobalToLocal
(
    void * globalAdrs    /* global address to convert */
)
```

**DESCRIPTION** This routine converts a global shared memory address *globalAdrs* to its corresponding local value. This routine does not verify that *globalAdrs* is really a valid global shared memory address.

All addresses stored in shared memory are global. Any access made to shared memory by the local CPU must be done using local addresses. This routine and `smObjLocalToGlobal()` are used to convert between these address types.

**AVAILABILITY** This routine is distributed as a component of the unbundled shared memory objects support option, VxMP.

**RETURNS** The local shared memory address pointed to by *globalAdrs*.

**SEE ALSO** `smObjLib`, `smObjLocalToGlobal()`

S

---

## smObjInit()

**NAME** smObjInit() – initialize a shared memory objects descriptor (VxMP Opt.)

**SYNOPSIS**

```
void smObjInit
(
    SM_OBJ_DESC * pSmObjDesc,      /* ptr to shared memory descriptor */
    SM_ANCHOR *  anchorLocalAdrs, /* shared memory anchor local adrs */
    int          ticksPerBeat,     /* cpu ticks per heartbeat */
    int          smObjMaxTries,    /* max no. of tries to obtain spinLock */
    int          intType,          /* interrupt method */
    int          intArg1,          /* interrupt argument #1 */
    int          intArg2,          /* interrupt argument #2 */
    int          intArg3           /* interrupt argument #3 */
)
```

**DESCRIPTION**

This routine initializes a shared memory descriptor. The descriptor must already be allocated in the CPU's local memory. Once the descriptor has been initialized by this routine, the CPU may attach itself to the shared memory area by calling **smObjAttach()**.

Only the shared memory descriptor itself is modified by this routine. No structures in shared memory are affected.

Parameters:

*pSmObjDesc*

The address of the shared memory descriptor to be initialized; this structure must be allocated before **smObjInit()** is called.

*anchorLocalAdrs*

The memory address by which the local CPU may access the shared memory anchor. This address may vary among CPUs in the system because of address offsets (particularly if the anchor is located in one CPU's dual-ported memory).

*ticksPerBeat*

Specifies the frequency of the shared memory anchor's heartbeat. The frequency is expressed in terms of how many CPU ticks on the local CPU correspond to one heartbeat period.

*smObjMaxTries*

Specifies the maximum number of tries to obtain access to an internal mutually exclusive data structure.

*intType, intArg1, intArg2, intArg3*

Allow a CPU to announce the method by which it is to be notified of shared memory events. See the manual entry for **if\_sm** for a discussion about interrupt types and their associated parameters.

This routine is called automatically when the component `INCLUDE_SM_OBJ` is included.

|                     |                                                                                                         |
|---------------------|---------------------------------------------------------------------------------------------------------|
| <b>AVAILABILITY</b> | This routine is distributed as a component of the unbundled shared memory objects support option, VxMP. |
| <b>RETURNS</b>      | N/A                                                                                                     |
| <b>SEE ALSO</b>     | <code>smObjLib</code> , <code>smObjSetup()</code> , <code>smObjAttach()</code>                          |

---

## smObjLibInit()

|                     |                                                                                                                                                  |
|---------------------|--------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>         | <code>smObjLibInit()</code> – install the shared memory objects facility (VxMP Opt.)                                                             |
| <b>SYNOPSIS</b>     | <b>STATUS</b> <code>smObjLibInit (void)</code>                                                                                                   |
| <b>DESCRIPTION</b>  | This routine installs the shared memory objects facility. It is called automatically when the component <code>INCLUDE_SM_OBJ</code> is included. |
| <b>AVAILABILITY</b> | This routine is distributed as a component of the unbundled shared memory objects support option, VxMP.                                          |
| <b>RETURNS</b>      | OK, or <code>ERROR</code> if the shared memory objects facility has already been installed.                                                      |
| <b>SEE ALSO</b>     | <code>smObjLib</code>                                                                                                                            |

---

## smObjLocalToGlobal()

|                    |                                                                                                                                                                                                           |
|--------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>        | <code>smObjLocalToGlobal()</code> – convert a local address to a global address (VxMP Opt.)                                                                                                               |
| <b>SYNOPSIS</b>    | <pre>void * smObjLocalToGlobal (     void * localAdrs      /* local address to convert */ )</pre>                                                                                                         |
| <b>DESCRIPTION</b> | This routine converts a local shared memory address <i>localAdrs</i> to its corresponding global value. This routine does not verify that <i>localAdrs</i> is really a valid local shared memory address. |

All addresses stored in shared memory are global. Any access made to shared memory by the local CPU must be done using local addresses. This routine and **smObjGlobalToLocal()** are used to convert between these address types.

**AVAILABILITY** This routine is distributed as a component of the unbundled shared memory objects support option, VxMP.

**RETURNS** The global shared memory address pointed to by *localAdrs*.

**SEE ALSO** **smObjLib**, **smObjGlobalToLocal()**

---

## smObjSetup()

**NAME** **smObjSetup()** – initialize the shared memory objects facility (VxMP Opt.)

**SYNOPSIS**

```
STATUS smObjSetup  
(  
    SM_OBJ_PARAMS * smObjParams /* setup parameters */  
)
```

**DESCRIPTION** This routine initializes the shared memory objects facility by filling the shared memory header. It must be called only once by the shared memory master CPU. It is called automatically only by the master CPU, when the component **INCLUDE\_SM\_OBJ** is included.

Any CPU on the system backplane can use the shared memory objects facility; however, the facility must first be initialized on the master CPU. Then before other CPUs are attached to the shared memory area by **smObjAttach()**, each must initialize its own shared memory objects descriptor using **smObjInit()**. This mechanism is similar to the one used by the shared memory network driver.

The *smObjParams* parameter is a pointer to a structure containing the values used to describe the shared memory objects setup. This structure is defined as follows in **smObjLib.h**:

```
typedef struct sm_obj_params /* setup parameters */  
{  
    BOOL allocatedPool; /* TRUE if shared memory pool is malloced */  
    SM_ANCHOR * pAnchor; /* shared memory anchor */  
    char * smObjFreeAdrs; /* start address of shared memory pool */  
    int smObjMemSize; /* memory size reserved for shared memory */  
    int maxCpus; /* max number of CPUs in the system */  
    int maxTasks; /* max number of tasks using smObj */  
}
```

```

int          maxSems;          /* max number of shared semaphores */
int          maxMsgQueues;     /* max number of shared message queues */
int          maxMemParts;     /* max number of shared memory partitions */
int          maxNames;        /* max number of names of shared objects */
} SM_OBJ_PARAMS;

```

- AVAILABILITY** This routine is distributed as a component of the unbundled shared memory objects support option, VxMP.
- RETURNS** OK, or ERROR if the shared memory pool cannot hold all the requested objects or the number of CPUs exceeds the maximum.
- ERRNO** S\_smObjLib\_TOO\_MANY\_CPU  
S\_smObjLib\_SHARED\_MEM\_TOO\_SMALL
- SEE ALSO** smObjLib, smObjInit(), smObjAttach()

---

## smObjShow()

- NAME** smObjShow() – display the current status of shared memory objects (VxMP Opt.)
- SYNOPSIS** STATUS smObjShow (void)
- DESCRIPTION** This routine displays useful information about the current status of shared memory objects facilities.

---

**WARNING:** The information returned by this routine is not static and may be obsolete by the time it is examined. This information is generally used for debugging purposes only.

---

**EXAMPLE**

```

-> smObjShow
Shared Mem Anchor Local Addr: 0x600.
Shared Mem Hdr Local Addr:   0xb1514.
Attached CPU :                5
Max Tries to Take Lock:      1
Shared Object Type    Current  Maximum  Available
-----
Tasks                 1         20        19
Binary Semaphores     8         30        20
Counting Semaphores   2         30        20
Messages Queues       3         10         7
Memory Partitions     1          4         3
Names in Database     16        100        84

```

|                     |                                                                                                         |
|---------------------|---------------------------------------------------------------------------------------------------------|
| <b>AVAILABILITY</b> | This routine is distributed as a component of the unbundled shared memory objects support option, VxMP. |
| <b>RETURNS</b>      | OK, or ERROR if no shared memory objects are initialized.                                               |
| <b>ERRNO</b>        | S_smObjLib_NOT_INITIALIZED                                                                              |
| <b>SEE ALSO</b>     | smObjShow, smObjLib                                                                                     |

---

## smObjTimeoutLogEnable()

|                     |                                                                                                                                                                           |
|---------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>         | <b>smObjTimeoutLogEnable()</b> – control logging of failed attempts to take a spin-lock (VxMP Opt.)                                                                       |
| <b>SYNOPSIS</b>     | <pre>void smObjTimeoutLogEnable (     BOOL timeoutLogEnable    /* TRUE to enable, FALSE to disable */ )</pre>                                                             |
| <b>DESCRIPTION</b>  | <p>This routine enables or disables the printing of a message when an attempt to take a shared memory spin-lock fails.</p> <p>By default, message logging is enabled.</p> |
| <b>AVAILABILITY</b> | This routine is distributed as a component of the unbundled shared memory objects support option, VxMP.                                                                   |
| <b>RETURNS</b>      | N/A                                                                                                                                                                       |
| <b>SEE ALSO</b>     | smObjLib                                                                                                                                                                  |

---

## sntpTimeGet()

|                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
|--------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>        | <code>sntpTimeGet()</code> – retrieve the current time from a remote source                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| <b>SYNOPSIS</b>    | <pre>STATUS sntpTimeGet (     char *          pServerAddr, /* server IP address or hostname */     u_int          timeout,     /* timeout interval in ticks */     struct timespec * pCurrTime /* storage for retrieved time value */ )</pre>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| <b>DESCRIPTION</b> | <p>This routine stores the current time as reported by an SNTP/NTP server in the location indicated by <i>pCurrTime</i>. The reported time is first converted to the elapsed time since January 1, 1970, 00:00, GMT, which is the base value used by UNIX systems. If <i>pServerAddr</i> is <code>NULL</code>, the routine listens for messages sent by an SNTP/NTP server in broadcast mode. Otherwise, this routine sends a request to the specified SNTP/NTP server and extracts the reported time from the reply. In either case, an error is returned if no message is received within the interval specified by <i>timeout</i>. Typically, SNTP/NTP servers operating in broadcast mode send update messages every 64 to 1024 seconds. An infinite timeout value is specified by <code>WAIT_FOREVER</code>.</p> |
| <b>RETURNS</b>     | <code>OK</code> , or <code>ERROR</code> if unsuccessful.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| <b>ERRNO</b>       | <code>S_sntpLib_INVALID_PARAMETER</code> , <code>S_sntpLib_INVALID_ADDRESS</code> , <code>S_sntpLib_TIMEOUT</code> , <code>S_sntpLib_SERVER_UNSYNC</code> , <code>S_sntpLib_VERSION_UNSUPPORTED</code>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
| <b>SEE ALSO</b>    | <code>sntpLib</code>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |

---

## sntpsClockSet()

**NAME** sntpsClockSet() – assign a routine to access the reference clock

**SYNOPSIS**

```
STATUS sntpsClockSet
(
    FUNCPTR pClockHookRtn    /* new interface to reference clock */
)
```

**DESCRIPTION** This routine installs a hook routine that is called to access the reference clock used by the SNTP server. This hook routine must use the following interface:

```
STATUS sntpsClockHook (int request, void *pBuffer);
```

The hook routine should copy one of three settings used by the server to construct outgoing NTP messages into *pBuffer* according to the value of the *request* parameter. If the requested setting is available, the installed routine should return **OK** (or **ERROR** otherwise).

This routine calls the given hook routine with the *request* parameter set to **SNTPS\_ID** to get the 32-bit reference identifier in the format specified in RFC 1769. It also calls the hook routine with *request* set to **SNTPS\_RESOLUTION** to retrieve a 32-bit value containing the clock resolution in nanoseconds. That value will be used to determine the 8-bit signed integer indicating the clock precision (according to the format specified in RFC 1769). Other library routines will set the *request* parameter to **SNTPS\_TIME** to retrieve the current 64-bit NTP timestamp from *pBuffer* in host byte order. The routine **sntpsNsecToFraction()** will convert a value in nanoseconds to the format required for the NTP fractional part.

### VXWORKS AE PROTECTION DOMAINS

Under VxWorks AE, you can call this function from within the kernel protection domain only. In addition, all arguments to this function can reference only that data which is valid in the kernel protection domain. This restriction does not apply under non-AE versions of VxWorks.

**RETURNS** **OK** or **ERROR**.

**ERRNO** N/A

**SEE ALSO** **sntpsLib**

---

## sntpConfigSet()

|                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
|--------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>        | <code>sntpConfigSet()</code> – change SNTP server broadcast settings                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
| <b>SYNOPSIS</b>    | <pre><b>STATUS</b> sntpConfigSet (     int    setting,          /* configuration option to change */     void * pValue           /* new value for parameter */ )</pre>                                                                                                                                                                                                                                                                                                                                                                             |
| <b>DESCRIPTION</b> | This routine alters the configuration of the SNTP server when operating in broadcast mode. A <i>setting</i> value of <code>SNTPS_DELAY</code> interprets the contents of <i>pValue</i> as the new 16-bit broadcast interval. When <i>setting</i> equals <code>SNTPS_ADDRESS</code> , <i>pValue</i> should provide the string representation of an IP broadcast or multicast address (for example, “224.0.1.1”). Any changed settings will take effect after the current broadcast interval is completed and the corresponding NTP message is sent. |
| <b>RETURNS</b>     | OK or ERROR.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
| <b>ERRNO</b>       | <code>S_sntpLib_INVALID_PARAMETER</code>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| <b>SEE ALSO</b>    | <code>sntpLib</code>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |

---

## sntpNsecToFraction()

|                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
|--------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>        | <code>sntpNsecToFraction()</code> – convert portions of a second to NTP format                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| <b>SYNOPSIS</b>    | <pre><b>ULONG</b> sntpNsecToFraction (     <b>ULONG</b> nsecs           /* nanoseconds to convert to binary fraction */ )</pre>                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| <b>DESCRIPTION</b> | This routine is provided for convenience in fulfilling an <code>SNTPS_TIME</code> request to the clock hook. It converts a value in nanoseconds to the fractional part of the NTP timestamp format. The routine is not designed to convert non-normalized values greater than or equal to one second. Although the NTP time format provides a precision of about 200 pico-seconds, rounding errors in the conversion process decrease the accuracy as the input value increases. In the worst case, only the 24 most significant bits are valid, which reduces the precision to tenths of a micro-second. |

**so()**

|                 |                                                   |
|-----------------|---------------------------------------------------|
| <b>RETURNS</b>  | Value for NTP fractional part in host-byte order. |
| <b>ERRNO</b>    | N/A                                               |
| <b>SEE ALSO</b> | <b>sntpsLib</b>                                   |

---

**so()**

|                    |                                                                                                                                                                                                                                                                                                                                                             |
|--------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>        | <b>so()</b> – single-step, but step over a subroutine                                                                                                                                                                                                                                                                                                       |
| <b>SYNOPSIS</b>    | <pre> <b>STATUS</b> <b>so</b>     (         <b>int</b> <b>task</b>                <i>/* task to step; 0 = use default */</i>     ) </pre>                                                                                                                                                                                                                   |
| <b>DESCRIPTION</b> | <p>This routine single-steps a task that is stopped at a breakpoint. However, if the next instruction is a JSR or BSR, <b>so()</b> breaks at the instruction following the subroutine call instead.</p> <p>To execute, enter:</p> <pre> -&gt; <b>so</b> [<b>task</b>] </pre> <p>If <i>task</i> is omitted or zero, the last task referenced is assumed.</p> |
| <b>SEE ALSO</b>    | <b>dbgLib</b> , <i>VxWorks Programmer's Guide: Target Shell</i> , <b>windsh</b> , <i>Tornado User's Guide: Shell</i>                                                                                                                                                                                                                                        |

---

**socket()**

|                 |                                                                                                                                                                                                                                                                                                                                                   |
|-----------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>     | <b>socket()</b> – open a socket                                                                                                                                                                                                                                                                                                                   |
| <b>SYNOPSIS</b> | <pre> <b>int</b> <b>socket</b>     (         <b>int</b> <b>domain</b>,                <i>/* address family (for example, AF_INET) */</i>         <b>int</b> <b>type</b>,                  <i>/* SOCK_STREAM, SOCK_DGRAM, or SOCK_RAW */</i>         <b>int</b> <b>protocol</b>              <i>/* socket protocol (usually 0) */</i>     ) </pre> |

**DESCRIPTION** This routine opens a socket and returns a socket descriptor. The socket descriptor is passed to the other socket routines to identify the socket. The socket descriptor is a standard I/O system file descriptor (*fd*) and can be used with the **close()**, **read()**, **write()**, and **ioctl()** routines.

Available socket types include:

**SOCK\_STREAM**  
Specifies a connection-based (stream) socket.

**SOCK\_DGRAM**  
Specifies a datagram (UDP) socket.

**SOCK\_RAW**  
Specifies a raw socket.

**RETURNS** A socket descriptor, or **ERROR**.

**SEE ALSO** **sockLib**

---

## sockUploadPathClose()

**NAME** **sockUploadPathClose()** – close the socket upload path (Windview)

**SYNOPSIS**

```
void sockUploadPathClose
(
    UPLOAD_ID upId          /* generic upload-path descriptor */
)
```

**DESCRIPTION** This routine closes the socket connection to the event receiver on the host.

**RETURNS** N/A

**SEE ALSO** **wvSockUploadPathLib**, **sockUploadPathCreate()**

## sockUploadPathCreate()

- NAME** sockUploadPathCreate() – establish an upload path to the host using a socket (Windview)
- SYNOPSIS**
- ```
UPLOAD_ID sockUploadPathCreate
(
    char * ipAddress,          /* server's hostname or IP address in */
                                /* .-notation */
    short port                 /* port number to bind to */
)
```
- DESCRIPTION** This routine initializes the TCP/IP connection to the host process that receives uploaded events. It can be retried if the connection attempt fails.
- RETURNS** The UPLOAD\_ID, or NULL if the connection cannot be completed or memory for the ID is not available.
- SEE ALSO** wvSockUploadPathLib, sockUploadPathClose()

---

## sockUploadPathLibInit()

- NAME** sockUploadPathLibInit() – initialize wvSockUploadPathLib library (Windview)
- SYNOPSIS**
- ```
STATUS sockUploadPathLibInit (void)
```
- DESCRIPTION** This routine initializes wvSockUploadPathLib by pulling in the routines in this file for use with WindView. It is called during system configuration from **usrWindview.c**.
- RETURN** OK.
- SEE ALSO** wvSockUploadPathLib

---

## sockUploadPathWrite()

- NAME** sockUploadPathWrite() – write to the socket upload path (Windview)
- SYNOPSIS**
- ```
int sockUploadPathWrite
(
    UPLOAD_ID upId,          /* generic upload-path descriptor */
    char *    pStart,        /* address of data to write */
    size_t    size           /* number of bytes of data at pStart */
)
```
- DESCRIPTION** This routine writes *size* bytes of data beginning at *pStart* to the upload path between the target and the event receiver on the host.
- RETURNS** The number of bytes written, or **ERROR**.
- SEE ALSO** wvSockUploadPathLib, sockUploadPathCreate()

---

## sp()

- NAME** sp() – spawn a task with default parameters
- SYNOPSIS**
- ```
int sp
(
    FUNCPTR func,           /* function to call */
    int    arg1,           /* first of nine args to pass to spawned task */
    int    arg2,
    int    arg3,
    int    arg4,
    int    arg5,
    int    arg6,
    int    arg7,
    int    arg8,
    int    arg9
)
```
- DESCRIPTION** This command spawns a specified function as a task with the following defaults:  
priority:  
100

**sprintf()**

stack size:

20,000 bytes

task ID:

highest not currently used

task options:

**VX\_FP\_TASK** - execute with floating-point coprocessor support.

task name:

A name of the form **tN** where N is an integer which increments as new tasks are spawned, e.g., **t1**, **t2**, **t3**, etc.

The task ID is displayed after the task is spawned.

This command is a short form of the underlying **taskSpawn()** routine, convenient for spawning tasks in which the default parameters are satisfactory. If the default parameters are unacceptable, **taskSpawn()** should be called directly.

**RETURNS** A task ID, or **ERROR** if the task cannot be spawned.

**SEE ALSO** **usrLib**, **taskLib**, **taskSpawn()**, *VxWorks Programmer's Guide: Target Shell*, **windsh**, *Tornado User's Guide: Shell*

**sprintf()**

**NAME** **sprintf()** – write a formatted string to a buffer (ANSI)

**SYNOPSIS**

```
int sprintf
(
    char *      buffer,      /* buffer to write to */
    const char * fmt,       /* format string */
    ...         /* optional arguments to format */
)
```

**DESCRIPTION** This routine copies a formatted string to a specified buffer, which is null-terminated. Its function and syntax are otherwise identical to **printf()**.

**RETURNS** The number of characters copied to *buffer*, not including the **NULL** terminator.

**SEE ALSO** **fiolib**, **printf()**, *American National Standard for Information Systems -Programming Language - C, ANSI X3.159-1989: Input/Output (stdio.h)*

---

## spy()

|                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
|--------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>        | <b>spy()</b> – begin periodic task activity reports                                                                                                                                                                                                                                                                                                                                                                                                                          |
| <b>SYNOPSIS</b>    | <pre>void spy (     int freq,                /* reporting freq in sec, 0 = default of 5 */     int ticksPerSec         /* interrupt clock freq, 0 = default of 100 */ )</pre>                                                                                                                                                                                                                                                                                                |
| <b>DESCRIPTION</b> | <p>This routine collects task activity data and periodically runs <b>spyReport()</b>. Data is gathered <i>ticksPerSec</i> times per second, and a report is made every <i>freq</i> seconds. If <i>freq</i> is zero, it defaults to 5 seconds. If <i>ticksPerSec</i> is omitted or zero, it defaults to 100.</p> <p>This routine spawns <b>spyTask()</b> to do the actual reporting.</p> <p>It is not necessary to call <b>spyClkStart()</b> before running <b>spy()</b>.</p> |
| <b>RETURNS</b>     | N/A                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| <b>SEE ALSO</b>    | <b>usrLib</b> , <b>spyLib</b> , <b>spyClkStart()</b> , <b>spyTask()</b> , <i>VxWorks Programmer's Guide: Target Shell</i>                                                                                                                                                                                                                                                                                                                                                    |

---

## spyClkStart()

|                    |                                                                                                                                                                                                                                                                 |
|--------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>        | <b>spyClkStart()</b> – start collecting task activity data                                                                                                                                                                                                      |
| <b>SYNOPSIS</b>    | <pre>STATUS spyClkStart (     int intsPerSec          /* timer interrupt freq, 0 = default of 100 */ )</pre>                                                                                                                                                    |
| <b>DESCRIPTION</b> | <p>This routine begins data collection by enabling the auxiliary clock interrupts at a frequency of <i>intsPerSec</i> interrupts per second. If <i>intsPerSec</i> is omitted or zero, the frequency will be 100. Data from previous collections is cleared.</p> |
| <b>RETURNS</b>     | <b>OK</b> , or <b>ERROR</b> if the CPU has no auxiliary clock, or if task create and delete hooks cannot be installed.                                                                                                                                          |
| <b>SEE ALSO</b>    | <b>usrLib</b> , <b>spyLib</b> , <b>sysAuxClkConnect()</b> , <i>VxWorks Programmer's Guide: Target Shell</i>                                                                                                                                                     |

---

## spyClkStop()

|                    |                                                                                                                              |
|--------------------|------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>        | <b>spyClkStop()</b> – stop collecting task activity data                                                                     |
| <b>SYNOPSIS</b>    | <code>void spyClkStop (void)</code>                                                                                          |
| <b>DESCRIPTION</b> | This routine disables the auxiliary clock interrupts. Data collected remains valid until the next <b>spyClkStart()</b> call. |
| <b>RETURNS</b>     | N/A                                                                                                                          |
| <b>SEE ALSO</b>    | <b>usrLib</b> , <b>spyLib</b> , <b>spyClkStart()</b> , <i>VxWorks Programmer's Guide: Target Shell</i>                       |

---

## spyHelp()

|                                        |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |                      |                 |                                        |                                                                     |                         |                      |                        |                                            |                      |                                  |                                       |                                                      |
|----------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------------------|-----------------|----------------------------------------|---------------------------------------------------------------------|-------------------------|----------------------|------------------------|--------------------------------------------|----------------------|----------------------------------|---------------------------------------|------------------------------------------------------|
| <b>NAME</b>                            | <b>spyHelp()</b> – display task monitoring help menu                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |                      |                 |                                        |                                                                     |                         |                      |                        |                                            |                      |                                  |                                       |                                                      |
| <b>SYNOPSIS</b>                        | <code>void spyHelp (void)</code>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |                      |                 |                                        |                                                                     |                         |                      |                        |                                            |                      |                                  |                                       |                                                      |
| <b>DESCRIPTION</b>                     | This routine displays a summary of <b>spyLib</b> utilities:<br><table><tr><td><code>spyHelp</code></td><td>Print this list</td></tr><tr><td><code>spyClkStart [ticksPerSec]</code></td><td>Start task activity monitor running at ticksPerSec ticks per second</td></tr><tr><td><code>spyClkStop</code></td><td>Stop collecting data</td></tr><tr><td><code>spyReport</code></td><td>Prints display of task activity statistics</td></tr><tr><td><code>spyStop</code></td><td>Stop collecting data and reports</td></tr><tr><td><code>spy [freq[,ticksPerSec]]</code></td><td>Start spyClkStart and do a report every freq seconds</td></tr></table><br><code>ticksPerSec</code> defaults to 100. <code>freq</code> defaults to 5 seconds. | <code>spyHelp</code> | Print this list | <code>spyClkStart [ticksPerSec]</code> | Start task activity monitor running at ticksPerSec ticks per second | <code>spyClkStop</code> | Stop collecting data | <code>spyReport</code> | Prints display of task activity statistics | <code>spyStop</code> | Stop collecting data and reports | <code>spy [freq[,ticksPerSec]]</code> | Start spyClkStart and do a report every freq seconds |
| <code>spyHelp</code>                   | Print this list                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |                      |                 |                                        |                                                                     |                         |                      |                        |                                            |                      |                                  |                                       |                                                      |
| <code>spyClkStart [ticksPerSec]</code> | Start task activity monitor running at ticksPerSec ticks per second                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |                      |                 |                                        |                                                                     |                         |                      |                        |                                            |                      |                                  |                                       |                                                      |
| <code>spyClkStop</code>                | Stop collecting data                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |                      |                 |                                        |                                                                     |                         |                      |                        |                                            |                      |                                  |                                       |                                                      |
| <code>spyReport</code>                 | Prints display of task activity statistics                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |                      |                 |                                        |                                                                     |                         |                      |                        |                                            |                      |                                  |                                       |                                                      |
| <code>spyStop</code>                   | Stop collecting data and reports                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |                      |                 |                                        |                                                                     |                         |                      |                        |                                            |                      |                                  |                                       |                                                      |
| <code>spy [freq[,ticksPerSec]]</code>  | Start spyClkStart and do a report every freq seconds                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |                      |                 |                                        |                                                                     |                         |                      |                        |                                            |                      |                                  |                                       |                                                      |
| <b>RETURNS</b>                         | N/A                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |                      |                 |                                        |                                                                     |                         |                      |                        |                                            |                      |                                  |                                       |                                                      |
| <b>SEE ALSO</b>                        | <b>usrLib</b> , <b>spyLib</b> , <i>VxWorks Programmer's Guide: Target Shell</i>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |                      |                 |                                        |                                                                     |                         |                      |                        |                                            |                      |                                  |                                       |                                                      |

---

## spyLibInit()

|                    |                                                                                                                                                                                                     |
|--------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>        | <b>spyLibInit()</b> – initialize task CPU utilization tool package                                                                                                                                  |
| <b>SYNOPSIS</b>    | <b>void spyLibInit (void)</b>                                                                                                                                                                       |
| <b>DESCRIPTION</b> | This routine initializes the task CPU utilization tool package. If the configuration macro <b>INCLUDE_SPY</b> is defined, it is called by the root task, <b>usrRoot()</b> , in <b>usrConfig.c</b> . |
| <b>RETURNS</b>     | N/A                                                                                                                                                                                                 |
| <b>SEE ALSO</b>    | <b>spyLib</b> , <b>usrLib</b>                                                                                                                                                                       |

---

## spyReport()

|                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
|--------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>        | <b>spyReport()</b> – display task activity data                                                                                                                                                                                                                                                                                                                                                                                                                                |
| <b>SYNOPSIS</b>    | <b>void spyReport (void)</b>                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| <b>DESCRIPTION</b> | This routine reports on data gathered at interrupt level for the amount of CPU time utilized by each task, the amount of time spent at interrupt level, the amount of time spent in the kernel, and the amount of idle time. Time is displayed in ticks and as a percentage, and the data is shown since both the last call to <b>spyClkStart()</b> and the last <b>spyReport()</b> . If no interrupts have occurred since the last <b>spyReport()</b> , nothing is displayed. |
| <b>RETURNS</b>     | N/A                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| <b>SEE ALSO</b>    | <b>usrLib</b> , <b>spyLib</b> , <b>spyClkStart()</b> , <i>VxWorks Programmer's Guide: Target Shell</i>                                                                                                                                                                                                                                                                                                                                                                         |

## spyStop()

|                    |                                                                                                                          |
|--------------------|--------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>        | <b>spyStop()</b> – stop spying and reporting                                                                             |
| <b>SYNOPSIS</b>    | <code>void spyStop (void)</code>                                                                                         |
| <b>DESCRIPTION</b> | This routine calls <b>spyClkStop()</b> . Any periodic reporting by <b>spyTask()</b> is terminated.                       |
| <b>RETURNS</b>     | N/A                                                                                                                      |
| <b>SEE ALSO</b>    | <b>usrLib</b> , <b>spyLib</b> , <b>spyClkStop()</b> , <b>spyTask()</b> , <i>VxWorks Programmer's Guide: Target Shell</i> |

---

## spyTask()

|                    |                                                                                                                                                                           |
|--------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>        | <b>spyTask()</b> – run periodic task activity reports                                                                                                                     |
| <b>SYNOPSIS</b>    | <pre>void spyTask (     int freq                /* reporting frequency, in seconds */ )</pre>                                                                             |
| <b>DESCRIPTION</b> | This routine is spawned as a task by <b>spy()</b> to provide periodic task activity reports. It prints a report, delays for the specified number of seconds, and repeats. |
| <b>RETURNS</b>     | N/A                                                                                                                                                                       |
| <b>SEE ALSO</b>    | <b>usrLib</b> , <b>spyLib</b> , <b>spy()</b> , <i>VxWorks Programmer's Guide: Target Shell</i>                                                                            |

---

## sqrt()

|                      |                                                                                                                                   |
|----------------------|-----------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>          | <code>sqrt()</code> – compute a non-negative square root (ANSI)                                                                   |
| <b>SYNOPSIS</b>      | <pre>double sqrt (     double x          /* value to compute the square root of */ )</pre>                                        |
| <b>DESCRIPTION</b>   | This routine computes the non-negative square root of $x$ in double precision. A domain error occurs if the argument is negative. |
| <b>INCLUDE FILES</b> | <code>math.h</code>                                                                                                               |
| <b>RETURNS</b>       | The double-precision square root of $x$ or 0 if $x$ is negative.                                                                  |
| <b>ERRNO</b>         | EDOM                                                                                                                              |
| <b>SEE ALSO</b>      | <code>ansiMath</code> , <code>mathALib</code>                                                                                     |

---

## sqrtf()

|                      |                                                                                           |
|----------------------|-------------------------------------------------------------------------------------------|
| <b>NAME</b>          | <code>sqrtf()</code> – compute a non-negative square root (ANSI)                          |
| <b>SYNOPSIS</b>      | <pre>float sqrtf (     float x          /* value to compute the square root of */ )</pre> |
| <b>DESCRIPTION</b>   | This routine returns the non-negative square root of $x$ in single precision.             |
| <b>INCLUDE FILES</b> | <code>math.h</code>                                                                       |
| <b>RETURNS</b>       | The single-precision square root of $x$ .                                                 |
| <b>SEE ALSO</b>      | <code>mathALib</code>                                                                     |

**sr()**

---

**sr()**

|                    |                                                                                                                                                                                                                                                                                                           |
|--------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>        | <b>sr()</b> – return the contents of the status register (68K, SH)                                                                                                                                                                                                                                        |
| <b>SYNOPSIS</b>    | <pre>int sr (     int taskId          /* task ID, 0 means default task */ )</pre>                                                                                                                                                                                                                         |
| <b>DESCRIPTION</b> | <p>This command extracts the contents of the status register from the TCB of a specified task. If <i>taskId</i> is omitted or zero, the last task referenced is assumed.</p> <p>For SH, similar routines are provided for all control registers (<b>gbr</b>, <b>vbr</b>): <b>gbr()</b>, <b>vbr()</b>.</p> |
| <b>RETURNS</b>     | The contents of the status register (or the requested control register).                                                                                                                                                                                                                                  |
| <b>SEE ALSO</b>    | <b>dbgArchLib</b> , <i>VxWorks Programmer's Guide: Target Shell</i>                                                                                                                                                                                                                                       |

---

**srand()**

|                      |                                                                                                                                                                                                                                                                                                                                                                    |
|----------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>          | <b>srand()</b> – reset the value of the seed used to generate random numbers (ANSI)                                                                                                                                                                                                                                                                                |
| <b>SYNOPSIS</b>      | <pre>void * srand (     uint_t seed          /* random number seed */ )</pre>                                                                                                                                                                                                                                                                                      |
| <b>DESCRIPTION</b>   | <p>This routine resets the seed value used by <b>rand()</b>. If <b>srand()</b> is then called with the same seed value, the sequence of pseudo-random numbers is repeated. If <b>rand()</b> is called before any calls to <b>srand()</b> have been made, the same sequence shall be generated as when <b>srand()</b> is first called with the seed value of 1.</p> |
| <b>INCLUDE FILES</b> | <b>stdlib.h</b>                                                                                                                                                                                                                                                                                                                                                    |
| <b>RETURNS</b>       | N/A                                                                                                                                                                                                                                                                                                                                                                |
| <b>SEE ALSO</b>      | <b>ansiStdlib</b> , <b>rand()</b>                                                                                                                                                                                                                                                                                                                                  |

---

## sscanf()

**NAME** `sscanf()` – read and convert characters from an ASCII string (ANSI)

**SYNOPSIS**

```
int sscanf
(
    const char * str,          /* string to scan */
    const char * fmt,         /* format string */
    ...                       /* optional arguments to format string */
)
```

**DESCRIPTION** This routine reads characters from the string *str*, interprets them according to format specifications in the string *fmt*, which specifies the admissible input sequences and how they are to be converted for assignment, using subsequent arguments as pointers to the objects to receive the converted input.

If there are insufficient arguments for the format, the behavior is undefined. If the format is exhausted while arguments remain, the excess arguments are evaluated but are otherwise ignored.

The format is a multibyte character sequence, beginning and ending in its initial shift state. The format is composed of zero or more directives: one or more white-space characters; an ordinary multibyte character (neither % nor a white-space character); or a conversion specification. Each conversion specification is introduced by the % character. After the %, the following appear in sequence:

- An optional assignment-suppressing character \*.
- An optional non-zero decimal integer that specifies the maximum field width.
- An optional **h**, **l** (ell) or **ll** (ell-ell) indicating the size of the

receiving object. The conversion specifiers **d**, **i**, and **n** should be preceded by **h** if the corresponding argument is a pointer to **short int** rather than a pointer to **int**, or by **l** if it is a pointer to **long int**, or by **ll** if it is a pointer to **long long int**. Similarly, the conversion specifiers **o**, **u**, and **x** shall be preceded by **h** if the corresponding argument is a pointer to **unsigned short int** rather than a pointer to **unsigned int**, or by **l** if it is a pointer to **unsigned long int**, or by **ll** if it is a pointer to **unsigned long long int**. Finally, the conversion specifiers **e**, **f**, and **g** shall be preceded by **l** if the corresponding argument is a pointer to **double** rather than a pointer to **float**. If a **h**, **l** or **ll** appears with any other conversion specifier, the behavior is undefined.

---

**WARNING:** ANSI C also specifies an optional **L** in some of the same contexts as **l** above, corresponding to a **long double** \* argument. However, the current release of the VxWorks libraries does not support **long double** data; using the optional **L** gives unpredictable results.

---

**sscanf()**

– A character that specifies the type of conversion to be applied. The valid conversion specifiers are described below.

The **sscanf()** routine executes each directive of the format in turn. If a directive fails, as detailed below, **sscanf()** returns. Failures are described as input failures (due to the unavailability of input characters), or matching failures (due to inappropriate input).

A directive composed of white-space character(s) is executed by reading input up to the first non-white-space character (which remains unread), or until no more characters can be read.

A directive that is an ordinary multibyte character is executed by reading the next characters of the stream. If one of the characters differs from one comprising the directive, the directive fails, and the differing and subsequent characters remain unread.

A directive that is a conversion specification defines a set of matching input sequences, as described below for each specifier. A conversion specification is executed in the following steps:

Input white-space characters (as specified by the **isspace()** function) are skipped, unless the specification includes a **l**, **c**, or **n** specifier.

An input item is read from the stream, unless the specification includes an **n** specifier. An input item is defined as the longest matching sequence of input characters, unless that exceeds a specified field width, in which case it is the initial subsequence of that length in the sequence. The first character, if any, after the input item remains unread. If the length of the input item is zero, the execution of the directive fails: this condition is a matching failure, unless an error prevented input from the stream, in which case it is an input failure.

Except in the case of a **%** specifier, the input item is converted to a type appropriate to the conversion specifier. If the input item is not a matching sequence, the execution of the directive fails: this condition is a matching failure. Unless assignment suppression was indicated by a **\***, the result of the conversion is placed in the object pointed to by the first argument following the *fmt* argument that has not already received a conversion result. If this object does not have an appropriate type, or if the result of the conversion cannot be represented in the space provided, the behavior is undefined.

The following conversion specifiers are valid:

**d**

Matches an optionally signed decimal integer whose format is the same as expected for the subject sequence of the **strtol()** function with the value 10 for the *base* argument. The corresponding argument should be a pointer to **int**.

**i**

Matches an optionally signed integer, whose format is the same as expected for the subject sequence of the **strtol()** function with the value 0 for the *base* argument. The corresponding argument should be a pointer to **int**.

- o** Matches an optionally signed octal integer, whose format is the same as expected for the subject sequence of the **strtoul()** function with the value 8 for the *base* argument. The corresponding argument should be a pointer to **unsigned int**.
- u** Matches an optionally signed decimal integer, whose format is the same as expected for the subject sequence of the **strtoul()** function with the value 10 for the *base* argument. The corresponding argument should be a pointer to **unsigned int**.
- x** Matches an optionally signed hexadecimal integer, whose format is the same as expected for the subject sequence of the **strtoul()** function with the value 16 for the *base* argument. The corresponding argument should be a pointer to **unsigned int**.
- e, f, g** Match an optionally signed floating-point number, whose format is the same as expected for the subject string of the **strtod()** function. The corresponding argument should be a pointer to **float**.
- s** Matches a sequence of non-white-space characters. The corresponding argument should be a pointer to the initial character of an array large enough to accept the sequence and a terminating null character, which will be added automatically.
- [** Matches a non-empty sequence of characters from a set of expected characters (the **scanset**). The corresponding argument should be a pointer to the initial character of an array large enough to accept the sequence and a terminating null character, which is added automatically. The conversion specifier includes all subsequent character in the format string, up to and including the matching right bracket (]). The characters between the brackets (the **scanlist**) comprise the scanset, unless the character after the left bracket is a circumflex (^) in which case the scanset contains all characters that do not appear in the scanlist between the circumflex and the right bracket. If the conversion specifier begins with "[]" or "[^]", the right bracket character is in the scanlist and the next right bracket character is the matching right bracket that ends the specification; otherwise the first right bracket character is the one that ends the specification.
- c** Matches a sequence of characters of the number specified by the field width (1 if no field width is present in the directive). The corresponding argument should be a pointer to the initial character of an array large enough to accept the sequence. No null character is added.
- p** Matches an implementation-defined set of sequences, which should be the same as the set of sequences that may be produced by the %p conversion of the **fprintf()** function. The corresponding argument should be a pointer to a pointer to **void**.

**sscanf()**

VxWorks defines its pointer input field to be consistent with pointers written by the **fprintf()** function (“0x” hexadecimal notation). If the input item is a value converted earlier during the same program execution, the pointer that results should compare equal to that value; otherwise the behavior of the %p conversion is undefined.

**n**

No input is consumed. The corresponding argument should be a pointer to **int** into which the number of characters read from the input stream so far by this call to **sscanf()** is written. Execution of a %n directive does not increment the assignment count returned when **sscanf()** completes execution.

**%**

Matches a single %; no conversion or assignment occurs. The complete conversion specification is %%.

If a conversion specification is invalid, the behavior is undefined.

The conversion specifiers **E**, **G**, and **X** are also valid and behave the same as **e**, **g**, and **x**, respectively.

If end-of-file is encountered during input, conversion is terminated. If end-of-file occurs before any characters matching the current directive have been read (other than leading white space, where permitted), execution of the current directive terminates with an input failure; otherwise, unless execution of the current directive is terminated with a matching failure, execution of the following directive (if any) is terminated with an input failure.

If conversion terminates on a conflicting input character, the offending input character is left unread in the input stream. Trailing white space (including new-line characters) is left unread unless matched by a directive. The success of literal matches and suppressed assignments is not directly determinable other than via the %n directive.

**INCLUDE FILES**

**fiolib.h**

**RETURNS**

The number of input items assigned, which can be fewer than provided for, or even zero, in the event of an early matching failure; or EOF if an input failure occurs before any conversion.

**SEE ALSO**

**fiolib**, **fscanf()**, **scanf()**, *American National Standard for Information Systems -Programming Language - C, ANSI X3.159-1989: Input/Output (stdio.h)*

---

## stackEntryIsBottom()

**NAME** `stackEntryIsBottom()` – test if an interface has no layers beneath it

**SYNOPSIS**

```
BOOL stackEntryIsBottom
(
    int index          /* interface to examine */
)
```

**DESCRIPTION** This routine returns **TRUE** if an interface has no layers beneath it. This helper function is not exported.

**RETURNS** **TRUE** if the interface is the bottom-most layer in a stack  
**FALSE** otherwise or on error

**SEE ALSO** `m2IfLib`

---

## stackEntryIsTop()

**NAME** `stackEntryIsTop()` – test if an ifStackTable interface has no layers above

**SYNOPSIS**

```
BOOL stackEntryIsTop
(
    int index          /* the interface to examine */
)
```

**DESCRIPTION** This routine returns **TRUE** if an interface is not below any other interface. That is, it returns **TRUE** if the given interface is topmost on a stack. This helper function is not exported.

**RETURNS** **TRUE** if interface is topmost  
**FALSE** otherwise or for errors

**SEE ALSO** `m2IfLib`

**stat()**

---

**stat()**

**NAME** `stat()` – get file status information using a pathname (POSIX)

**SYNOPSIS**

```
STATUS stat
(
    char *      name,          /* name of file to check */
    struct stat * pStat       /* pointer to stat structure */
)
```

**DESCRIPTION** This routine obtains various characteristics of a file (or directory). This routine is equivalent to `fstat()`, except that the *name* of the file is specified, rather than an open file descriptor.

The *pStat* parameter is a pointer to a `stat` structure (defined in `stat.h`). This structure must have already been allocated before this routine is called.

---

**NOTE:** When used with `netDrv` devices (FTP or RSH), `stat()` returns the size of the file and always sets the mode to regular; `stat()` does not distinguish between files, directories, links, etc.

---

On return, the fields in the `stat` structure are updated to reflect the characteristics of the file.

**RETURNS** OK or ERROR.

**SEE ALSO** `dirLib`, `fstat()`, `ls()`

---

**statfs()**

**NAME** `statfs()` – get file status information using a pathname (POSIX)

**SYNOPSIS**

```
STATUS statfs
(
    char *      name,          /* name of file to check */
    struct statfs * pStat     /* pointer to statfs structure */
)
```

**DESCRIPTION** This routine obtains various characteristics of a file system. This routine is equivalent to `fstatfs()`, except that the *name* of the file is specified, rather than an open file descriptor.

The *pStat* parameter is a pointer to a **statfs** structure (defined in **stat.h**). This structure must have already been allocated before this routine is called.

Upon return, the fields in the **statfs** structure are updated to reflect the characteristics of the file.

**RETURNS** OK or ERROR.

**SEE ALSO** `dirLib`, `fstatfs()`, `ls()`

---

## stdioFp()

**NAME** `stdioFp()` – return the standard input/output/error FILE of the current task

**SYNOPSIS**

```
FILE * stdioFp
(
    int stdFd                /* fd of standard FILE to return (0,1,2) */
)
```

**DESCRIPTION** This routine returns the specified standard FILE structure address of the current task. It is provided primarily to give access to standard input, standard output, and standard error from the shell, where the usual **stdin**, **stdout**, **stderr** macros cannot be used.

**INCLUDE FILES** `stdio.h`

**RETURNS** The standard FILE structure address of the specified file descriptor, for the current task.

**SEE ALSO** `ansiStdio`

---

## stdioInit()

**NAME** `stdioInit()` – initialize standard I/O support

**SYNOPSIS**

```
STATUS stdioInit (void)
```

**DESCRIPTION** This routine installs standard I/O support. It must be called before using **stdio** buffering. If **INCLUDE\_STDIO** is defined in **configAll.h**, it is called automatically by the root task `usrRoot()` in `usrConfig.c`.

**RETURNS** OK, or **ERROR** if the standard I/O facilities cannot be installed.

**SEE ALSO** **ansiStdio**

---

## stdioShow()

**NAME** **stdioShow()** – display file pointer internals

**SYNOPSIS**

```
STATUS stdioShow  
(  
    FILE * fp,                /* stream */  
    int   level              /* level */  
)
```

**DESCRIPTION** This routine displays information about a specified stream.

**RETURNS** OK, or **ERROR** if the file pointer is invalid.

**SEE ALSO** **ansiStdio**

---

## stdioShowInit()

**NAME** **stdioShowInit()** – initialize the standard I/O show facility

**SYNOPSIS** **STATUS** **stdioShowInit** (**void**)

**DESCRIPTION** This routine links the file pointer show routine into the VxWorks system. It is called automatically when this show facility is configured into VxWorks using either of the following methods:

- If you use the configuration header files, define **INCLUDE\_SHOW\_ROUTINES** in **config.h**.
- If you use the Tornado project facility, select **INCLUDE\_STDIO\_SHOW**.

**RETURNS** OK, or **ERROR** if an error occurs installing the file pointer show routine.

**SEE ALSO** **ansiStdio**

---

## strcat()

|                      |                                                                                                                                                            |
|----------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>          | <b>strcat()</b> – concatenate one string to another (ANSI)                                                                                                 |
| <b>SYNOPSIS</b>      | <pre>char * strcat (     char *      destination, /* string to be appended to */     const char * append     /* string to append to destination */ )</pre> |
| <b>DESCRIPTION</b>   | This routine appends a copy of string <i>append</i> to the end of string <i>destination</i> . The resulting string is null-terminated.                     |
| <b>INCLUDE FILES</b> | <b>string.h</b>                                                                                                                                            |
| <b>RETURNS</b>       | A pointer to <i>destination</i> .                                                                                                                          |
| <b>SEE ALSO</b>      | <b>ansiString</b>                                                                                                                                          |

---

## strchr()

|                      |                                                                                                                                                         |
|----------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>          | <b>strchr()</b> – find the first occurrence of a character in a string (ANSI)                                                                           |
| <b>SYNOPSIS</b>      | <pre>char * strchr (     const char * s,          /* string in which to search */     int         c           /* character to find in string */ )</pre> |
| <b>DESCRIPTION</b>   | This routine finds the first occurrence of character <i>c</i> in string <i>s</i> . The terminating null is considered to be part of the string.         |
| <b>INCLUDE FILES</b> | <b>string.h</b>                                                                                                                                         |
| <b>RETURNS</b>       | The address of the located character, or NULL if the character is not found.                                                                            |
| <b>SEE ALSO</b>      | <b>ansiString</b>                                                                                                                                       |

## strcmp()

|                      |                                                                                                                                                                       |
|----------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>          | <b>strcmp()</b> – compare two strings lexicographically (ANSI)                                                                                                        |
| <b>SYNOPSIS</b>      | <pre>int strcmp (     const char * s1,          /* string to compare */     const char * s2          /* string to compare s1 to */ )</pre>                            |
| <b>DESCRIPTION</b>   | This routine compares string <i>s1</i> to string <i>s2</i> lexicographically.                                                                                         |
| <b>INCLUDE FILES</b> | <b>string.h</b>                                                                                                                                                       |
| <b>RETURNS</b>       | An integer greater than, equal to, or less than 0, according to whether <i>s1</i> is lexicographically greater than, equal to, or less than <i>s2</i> , respectively. |
| <b>SEE ALSO</b>      | <b>ansiString</b>                                                                                                                                                     |

---

## strcoll()

|                      |                                                                                                                                                                                                                      |
|----------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>          | <b>strcoll()</b> – compare two strings as appropriate to LC_COLLATE (ANSI)                                                                                                                                           |
| <b>SYNOPSIS</b>      | <pre>int strcoll (     const char * s1,          /* string 1 */     const char * s2          /* string 2 */ )</pre>                                                                                                  |
| <b>DESCRIPTION</b>   | This routine compares two strings, both interpreted as appropriate to the LC_COLLATE category of the current locale.                                                                                                 |
| <b>INCLUDE FILES</b> | <b>string.h</b>                                                                                                                                                                                                      |
| <b>RETURNS</b>       | An integer greater than, equal to, or less than zero, according to whether string <i>s1</i> is greater than, equal to, or less than string <i>s2</i> when both are interpreted as appropriate to the current locale. |
| <b>SEE ALSO</b>      | <b>ansiString</b>                                                                                                                                                                                                    |

---

## strncpy()

|                      |                                                                                                                                           |
|----------------------|-------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>          | <b>strncpy()</b> – copy one string to another (ANSI)                                                                                      |
| <b>SYNOPSIS</b>      | <pre>char * strncpy (     char *      s1,          /* string to copy to */     const char * s2          /* string to copy from */ )</pre> |
| <b>DESCRIPTION</b>   | This routine copies string <i>s2</i> (including EOS) to string <i>s1</i> .                                                                |
| <b>INCLUDE FILES</b> | <b>string.h</b>                                                                                                                           |
| <b>RETURNS</b>       | A pointer to <i>s1</i> .                                                                                                                  |
| <b>SEE ALSO</b>      | <b>ansiString</b>                                                                                                                         |

---

## strcspn()

|                      |                                                                                                                                                             |
|----------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>          | <b>strcspn()</b> – return the string length up to the first character from a given set (ANSI)                                                               |
| <b>SYNOPSIS</b>      | <pre>size_t strcspn (     const char * s1,          /* string to search */     const char * s2          /* set of characters to look for in s1 */ )</pre>   |
| <b>DESCRIPTION</b>   | This routine computes the length of the maximum initial segment of string <i>s1</i> that consists entirely of characters not included in string <i>s2</i> . |
| <b>INCLUDE FILES</b> | <b>string.h</b>                                                                                                                                             |
| <b>RETURNS</b>       | The length of the string segment.                                                                                                                           |
| <b>SEE ALSO</b>      | <b>ansiString, strpbrk(), strspn()</b>                                                                                                                      |

## strerror()

|                    |                                                                                                                                                                                                                                                      |
|--------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>        | <b>strerror()</b> – map an error number to an error string (ANSI)                                                                                                                                                                                    |
| <b>SYNOPSIS</b>    | <pre>char * strerror (     int  errcode          /* error code */ )</pre>                                                                                                                                                                            |
| <b>DESCRIPTION</b> | <p>This routine maps the error number in <i>errcode</i> to an error message string. It returns a pointer to a static buffer that holds the error string.</p> <p>This routine is not reentrant. For a reentrant version, see <b>strerror_r()</b>.</p> |
| <b>INCLUDE</b>     | <b>string.h</b>                                                                                                                                                                                                                                      |
| <b>RETURNS</b>     | A pointer to the buffer that holds the error string.                                                                                                                                                                                                 |
| <b>SEE ALSO</b>    | <b>ansiString, strerror_r()</b>                                                                                                                                                                                                                      |

---

## strerror\_r()

|                      |                                                                                                                                                                                              |
|----------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>          | <b>strerror_r()</b> – map an error number to an error string (POSIX)                                                                                                                         |
| <b>SYNOPSIS</b>      | <pre>STATUS strerror_r (     int  errcode,          /* error number */     char * buffer          /* string buffer */ )</pre>                                                                |
| <b>DESCRIPTION</b>   | <p>This call maps the error code in <i>errcode</i> to an error message string which it stores in <i>buffer</i>.</p> <p>This routine is the POSIX reentrant version of <b>strerror()</b>.</p> |
| <b>INCLUDE FILES</b> | <b>string.h</b>                                                                                                                                                                              |
| <b>RETURNS</b>       | OK or ERROR.                                                                                                                                                                                 |
| <b>SEE ALSO</b>      | <b>ansiString, strerror()</b>                                                                                                                                                                |

---

## strftime()

**NAME** `strftime()` – convert broken-down time into a formatted string (ANSI)

**SYNOPSIS**

```
size_t strftime
(
    char *          s,      /* string array */
    size_t         n,      /* maximum size of array */
    const char *    format, /* format of output string */
    const struct tm * tptr  /* broken-down time */
)
```

**DESCRIPTION** This routine formats the broken-down time in *tptr* based on the conversion specified in the string *format*, and places the result in the string *s*.

The format is a multibyte character sequence, beginning and ending in its initial state. The *format* string consists of zero or more conversion specifiers and ordinary multibyte characters. A conversion specifier consists of a % character followed by a character that determines the behavior of the conversion. All ordinary multibyte characters (including the terminating NULL character) are copied unchanged to the array. If copying takes place between objects that overlap, the behavior is undefined. No more than *n* characters are placed into the array.

Each conversion specifier is replaced by appropriate characters as described in the following list. The appropriate characters are determined by the LC\_TIME category of the current locale and by the values contained in the structure pointed to by *tptr*.

- %a  
the locale's abbreviated weekday name.
- %A  
the locale's full weekday name.
- %b  
the locale's abbreviated month name.
- %B  
the locale's full month name.
- %c  
the locale's appropriate date and time representation.
- %d  
the day of the month as decimal number (01-31).
- %H  
the hour (24-hour clock) as a decimal number (00-23).

**%I**  
the hour (12-hour clock) as a decimal number (01-12).

**%j**  
the day of the year as decimal number (001-366).

**%m**  
the month as a decimal number (01-12).

**%M**  
the minute as a decimal number (00-59).

**%P**  
the locale's equivalent of the AM/PM designations associated with a 12-hour clock.

**%S**  
the second as a decimal number (00-59).

**%U**  
the week number of the year (first Sunday as first day of week 1) as a decimal number (00-53).

**%w**  
the weekday as a decimal number (0-6), where Sunday is 0.

**%W**  
the week number of the year (the first Monday as the first day of week 1) as a decimal number (00-53).

**%x**  
the locale's appropriate date representation.

**%X**  
the locale's appropriate time representation.

**%y**  
the year without century as a decimal number (00-99).

**%Y**  
the year with century as a decimal number.

**%Z**  
the time zone name or abbreviation, or by no characters if no time zone is determinable.

**%%**  
%.

For any other conversion specifier, the behavior is undefined.

**INCLUDE FILES**    **time.h**

**RETURNS** The number of characters in *s*, not including the terminating null character -- or zero if the number of characters in *s*, including the null character, is more than *n* (in which case the contents of *s* are indeterminate).

**SEE ALSO** `ansiTime`

---

## strlen()

**NAME** `strlen()` – determine the length of a string (ANSI)

**SYNOPSIS**

```
size_t strlen
(
    const char * s          /* string */
)
```

**DESCRIPTION** This routine returns the number of characters in *s*, not including EOS.

**INCLUDE FILES** `string.h`

**RETURNS** The number of non-null characters in the string.

**SEE ALSO** `ansiString`

---

## strncat()

**NAME** `strncat()` – concatenate characters from one string to another (ANSI)

**SYNOPSIS**

```
char * strncat
(
    char *      dst,          /* string to append to */
    const char * src,        /* string to append */
    size_t     n              /* max no. of characters to append */
)
```

**DESCRIPTION** This routine appends up to *n* characters from string *src* to the end of string *dst*.

**INCLUDE FILES** `string.h`

**RETURNS** A pointer to the null-terminated string *s1*.

**SEE ALSO** [ansiString](#)

---

## strncmp()

**NAME** `strncmp()` – compare the first *n* characters of two strings (ANSI)

**SYNOPSIS**

```
int strncmp
(
    const char * s1,          /* string to compare */
    const char * s2,          /* string to compare s1 to */
    size_t      n            /* max no. of characters to compare */
)
```

**DESCRIPTION** This routine compares up to *n* characters of string *s1* to string *s2*lexicographically.

**INCLUDE FILES** `string.h`

**RETURNS** An integer greater than, equal to, or less than 0, according to whether *s1* is lexicographically greater than, equal to, or less than *s2*, respectively.

**SEE ALSO** [ansiString](#)

---

## strncpy()

**NAME** `strncpy()` – copy characters from one string to another (ANSI)

**SYNOPSIS**

```
char *strncpy
(
    char *      s1,          /* string to copy to */
    const char * s2,          /* string to copy from */
    size_t      n            /* max no. of characters to copy */
)
```

|                      |                                                                                                                                                                                                                                                                                        |
|----------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>DESCRIPTION</b>   | This routine copies <i>n</i> characters from string <i>s2</i> to string <i>s1</i> . If <i>n</i> is greater than the length of <i>s2</i> , nulls are added to <i>s1</i> . If <i>n</i> is less than or equal to the length of <i>s2</i> , the target string will not be null-terminated. |
| <b>INCLUDE FILES</b> | <b>string.h</b>                                                                                                                                                                                                                                                                        |
| <b>RETURNS</b>       | A pointer to <i>s1</i> .                                                                                                                                                                                                                                                               |
| <b>SEE ALSO</b>      | <b>ansiString</b>                                                                                                                                                                                                                                                                      |

---

## strpbrk()

|                      |                                                                                                                                                           |
|----------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>          | <b>strpbrk()</b> – find the first occurrence in a string of a character from a given set (ANSI)                                                           |
| <b>SYNOPSIS</b>      | <pre>char * strpbrk (     const char * s1,          /* string to search */     const char * s2          /* set of characters to look for in s1 */ )</pre> |
| <b>DESCRIPTION</b>   | This routine locates the first occurrence in string <i>s1</i> of any character from string <i>s2</i> .                                                    |
| <b>INCLUDE FILES</b> | <b>string.h</b>                                                                                                                                           |
| <b>RETURNS</b>       | A pointer to the character found in <i>s1</i> , or NULL if no character from <i>s2</i> occurs in <i>s1</i> .                                              |
| <b>SEE ALSO</b>      | <b>ansiString, strcspn()</b>                                                                                                                              |

---

## strrchr()

|                 |                                                                                                                                            |
|-----------------|--------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>     | <b>strrchr()</b> – find the last occurrence of a character in a string (ANSI)                                                              |
| <b>SYNOPSIS</b> | <pre>char * strrchr (     const char * s,          /* string to search */     int          c           /* character to look for */ )</pre> |

|                      |                                                                                                                                                          |
|----------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>DESCRIPTION</b>   | This routine locates the last occurrence of <i>c</i> in the string pointed to by <i>s</i> . The terminating null is considered to be part of the string. |
| <b>INCLUDE FILES</b> | <code>string.h</code>                                                                                                                                    |
| <b>RETURNS</b>       | A pointer to the last occurrence of the character, or NULL if the character is not found.                                                                |
| <b>SEE ALSO</b>      | <code>ansiString</code>                                                                                                                                  |

---

## **strspn()**

**NAME** `strspn()` – return the string length up to the first character not in a given set (ANSI)

**SYNOPSIS**

```
size_t strspn
(
    const char * s,          /* string to search */
    const char * sep        /* set of characters to look for in s */
)
```

**DESCRIPTION** This routine computes the length of the maximum initial segment of string *s* that consists entirely of characters from the string *sep*.

**INCLUDE FILES** `string.h`

**RETURNS** The length of the string segment.

**SEE ALSO** `ansiString`, `strcspn()`

---

## **strstr()**

**NAME** `strstr()` – find the first occurrence of a substring in a string (ANSI)

**SYNOPSIS**

```
char * strstr
(
    const char * s,          /* string to search */
    const char * find       /* substring to look for */
)
```

|                      |                                                                                                                                                                   |
|----------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>DESCRIPTION</b>   | This routine locates the first occurrence in string <i>s</i> of the sequence of characters (excluding the terminating null character) in the string <i>find</i> . |
| <b>INCLUDE FILES</b> | <code>string.h</code>                                                                                                                                             |
| <b>RETURNS</b>       | A pointer to the located substring, or <i>s</i> if <i>find</i> points to a zero-length string, or <code>NULL</code> if the string is not found.                   |
| <b>SEE ALSO</b>      | <code>ansiString</code>                                                                                                                                           |

---

## strtod()

**NAME** `strtod()` – convert the initial portion of a string to a double (ANSI)

**SYNOPSIS**

```
double strtod
(
    const char * s,          /* string to convert */
    char * *   endptr       /* ptr to final string */
)
```

**DESCRIPTION** This routine converts the initial portion of a specified string *s* to a double. First, it decomposes the input string into three parts: an initial, possibly empty, sequence of white-space characters (as specified by the `isspace()` function); a subject sequence resembling a floating-point constant; and a final string of one or more unrecognized characters, including the terminating null character of the input string. Then, it attempts to convert the subject sequence to a floating-point number, and returns the result.

The expected form of the subject sequence is an optional plus or minus decimal-point character, then an optional exponent part but no floating suffix. The subject sequence is defined as the longest initial subsequence of the input string, starting with the first non-white-space character, that is of the expected form. The subject sequence contains no characters if the input string is empty or consists entirely of white space, or if the first non-white-space character is other than a sign, a digit, or a decimal-point character.

If the subject sequence has the expected form, the sequence of characters starting with the first digit or the decimal-point character (whichever occurs first) is interpreted as a floating constant, except that the decimal-point character is used in place of a period, and that if neither an exponent part nor a decimal-point character appears, a decimal point is assumed to follow the last digit in the string. If the subject sequence begins with a minus sign, the value resulting from the conversion is negated. A pointer to the final string is stored in the object pointed to by *endptr*, provided that *endptr* is not a null pointer.

**strtok()**

In other than the “C” locale, additional implementation-defined subject sequence forms may be accepted. VxWorks supports only the “C” locale.

If the subject sequence is empty or does not have the expected form, no conversion is performed; the value of *s* is stored in the object pointed to by *endptr*, provided that *endptr* is not a null pointer.

**INCLUDE FILES**     **stdlib.h**

**RETURNS**           The converted value, if any. If no conversion could be performed, it returns zero. If the correct value is outside the range of representable values, it returns plus or minus **HUGE\_VAL** (according to the sign of the value), and stores the value of the macro **ERANGE** in **errno**. If the correct value would cause underflow, it returns zero and stores the value of the macro **ERANGE** in **errno**.

**SEE ALSO**           **ansiStdlib**

---

## strtok()

**NAME**               **strtok()** – break down a string into tokens (ANSI)

**SYNOPSIS**

```
char * strtok
(
    char *      string,      /* string */
    const char * separator  /* separator indicator */
)
```

**DESCRIPTION**      A sequence of calls to this routine breaks the string *string* into a sequence of tokens, each of which is delimited by a character from the string *separator*. The first call in the sequence has *string* as its first argument, and is followed by calls with a null pointer as their first argument. The separator string may be different from call to call.

The first call in the sequence searches *string* for the first character that is not contained in the current separator string. If the character is not found, there are no tokens in *string* and **strtok()** returns a null pointer. If the character is found, it is the start of the first token.

**strtok()** then searches from there for a character that is contained in the current separator string. If the character is not found, the current token expands to the end of the string pointed to by *string*, and subsequent searches for a token will return a null pointer. If the character is found, it is overwritten by a null character, which terminates the current token. **strtok()** saves a pointer to the following character, from which the next search for a token will start. (Note that because the separator character is overwritten by a null character, the input string is modified as a result of this call.)

Each subsequent call, with a null pointer as the value of the first argument, starts searching from the saved pointer and behaves as described above.

The implementation behaves as if `strtok()` is called by no library functions.

|                      |                                                                                                   |
|----------------------|---------------------------------------------------------------------------------------------------|
| <b>REENTRANCY</b>    | This routine is not reentrant; the reentrant form is <code>strtok_r()</code> .                    |
| <b>INCLUDE FILES</b> | <code>string.h</code>                                                                             |
| <b>RETURNS</b>       | A pointer to the first character of a token, or a <code>NULL</code> pointer if there is no token. |
| <b>SEE ALSO</b>      | <code>ansiString</code> , <code>strtok_r()</code>                                                 |

---

## strtok\_r()

**NAME** `strtok_r()` – break down a string into tokens (reentrant) (POSIX)

**SYNOPSIS**

```
char * strtok_r
(
    char *      string,      /* string to break into tokens */
    const char * separators, /* the separators */
    char * *    ppLast      /* pointer to serve as string index */
)
```

**DESCRIPTION** This routine considers the null-terminated string *string* as a sequence of zero or more text tokens separated by spans of one or more characters from the separator string *separators*. The argument *ppLast* points to a user-provided pointer which in turn points to the position within *string* at which scanning should begin.

In the first call to this routine, *string* points to a null-terminated string; *separators* points to a null-terminated string of separator characters; and *ppLast* points to a `NULL` pointer. The function returns a pointer to the first character of the first token, writes a null character into *string* immediately following the returned token, and updates the pointer to which *ppLast* points so that it points to the first character following the null written into *string*. (Note that because the separator character is overwritten by a null character, the input string is modified as a result of this call.)

In subsequent calls *string* must be a `NULL` pointer and *ppLast* must be unchanged so that subsequent calls will move through the string *string*, returning successive tokens until no tokens remain. The separator string *separators* may be different from call to call. When no token remains in *string*, a `NULL` pointer is returned.

**INCLUDE FILES** `string.h`

**strtol()**

**RETURNS** A pointer to the first character of a token, or a NULL pointer if there is no token.

**SEE ALSO** `ansiString`, `strtok()`

**strtol()**

**NAME** `strtol()` – convert a string to a long integer (ANSI)

**SYNOPSIS**

```
long strtol
(
    const char * nptr,      /* string to convert */
    char * *   endptr,     /* ptr to final string */
    int        base       /* radix */
)
```

**DESCRIPTION** This routine converts the initial portion of a string *nptr* to **long int** representation. First, it decomposes the input string into three parts: an initial, possibly empty, sequence of white-space characters (as specified by `isspace()`); a subject sequence resembling an integer represented in some radix determined by the value of *base*; and a final string of one or more unrecognized characters, including the terminating NULL character of the input string. Then, it attempts to convert the subject sequence to an integer number, and returns the result.

If the value of *base* is zero, the expected form of the subject sequence is that of an integer constant, optionally preceded by a plus or minus sign, but not including an integer suffix. If the value of *base* is between 2 and 36, the expected form of the subject sequence is a sequence of letters and digits representing an integer with the radix specified by *base* optionally preceded by a plus or minus sign, but not including an integer suffix. The letters from a (or A) through to z (or Z) are ascribed the values 10 to 35; only letters whose ascribed values are less than *base* are permitted. If the value of *base* is 16, the characters 0x or 0X may optionally precede the sequence of letters and digits, following the sign if present.

The subject sequence is defined as the longest initial subsequence of the input string, starting with the first non-white-space character, that is of the expected form. The subject sequence contains no characters if the input string is empty or consists entirely of white space, or if the first non-white-space character is other than a sign or a permissible letter or digit.

If the subject sequence has the expected form and the value of *base* is zero, the sequence of characters starting with the first digit is interpreted as an integer constant. If the subject sequence has the expected form and the value of *base* is between 2 and 36, it is used as the *base* for conversion, ascribing to each letter its value as given above. If the subject sequence

begins with a minus sign, the value resulting from the conversion is negated. A pointer to the final string is stored in the object pointed to by *endptr*, provided that *endptr* is not a NULL pointer.

In other than the “C” locale, additional implementation-defined subject sequence forms may be accepted. VxWorks supports only the “C” locale; it assumes that the upper- and lower-case alphabets and digits are each contiguous.

If the subject sequence is empty or does not have the expected form, no conversion is performed; the value of *nptr* is stored in the object pointed to by *endptr*, provided that *endptr* is not a NULL pointer.

**INCLUDE FILES**     **stdlib.h**

**RETURNS**           The converted value, if any. If no conversion could be performed, it returns zero. If the correct value is outside the range of representable values, it returns **LONG\_MAX** or **LONG\_MIN** (according to the sign of the value), and stores the value of the macro **ERANGE** in **errno**.

**SEE ALSO**           **ansiStdlib**

---

## strtoul()

**NAME**               **strtoul()** – convert a string to an unsigned long integer (ANSI)

**SYNOPSIS**           **ulong\_t strtoul**

```
(
  const char * nptr,          /* string to convert */
  char * *   endptr,         /* ptr to final string */
  int        base            /* radix */
)
```

**DESCRIPTION**       This routine converts the initial portion of a string *nptr* to **unsigned long int** representation. First, it decomposes the input string into three parts: an initial, possibly empty, sequence of white-space characters (as specified by **isspace()**); a subject sequence resembling an unsigned integer represented in some radix determined by the value *base*; and a final string of one or more unrecognized characters, including the terminating null character of the input string. Then, it attempts to convert the subject sequence to an unsigned integer, and returns the result.

If the value of *base* is zero, the expected form of the subject sequence is that of an integer constant, optionally preceded by a plus or minus sign, but not including an integer suffix. If the value of *base* is between 2 and 36, the expected form of the subject sequence is a

**strtoul()**

sequence of letters and digits representing an integer with the radix specified by letters from a (or A) through z (or Z) which are ascribed the values 10 to 35; only letters whose ascribed values are less than *base* are permitted. If the value of *base* is 16, the characters 0x or 0X may optionally precede the sequence of letters and digits, following the sign if present.

The subject sequence is defined as the longest initial subsequence of the input string, starting with the first non-white-space character, that is of the expected form. The subject sequence contains no characters if the input string is empty or consists entirely of white space, or if the first non-white-space character is other than a sign or a permissible letter or digit.

If the subject sequence has the expected form and the value of *base* is zero, the sequence of characters starting with the first digit is interpreted as an integer constant. If the subject sequence has the expected form and the value of *base* is between 2 and 36, it is used as the *base* for conversion, ascribing to each letter its value as given above. If the subject sequence begins with a minus sign, the value resulting from the conversion is negated. A pointer to the final string is stored in the object pointed to by *endptr*, provided that *endptr* is not a null pointer.

In other than the "C" locale, additional implementation-defined subject sequence forms may be accepted. VxWorks supports only the "C" locale; it assumes that the upper- and lower-case alphabets and digits are each contiguous.

If the subject sequence is empty or does not have the expected form, no conversion is performed; the value of *nptr* is stored in the object pointed to by *endptr*, provided that *endptr* is not a null pointer.

**INCLUDE FILES**     **stdlib.h**

**RETURNS**           The converted value, if any. If no conversion could be performed it returns zero. If the correct value is outside the range of representable values, it returns **ULONG\_MAX**, and stores the value of the macro **ERANGE** in **errno**.

**SEE ALSO**           **ansiStdlib**

---

## strxfrm()

|                      |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
|----------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>          | <code>strxfrm()</code> – transform up to <i>n</i> characters of <i>s2</i> into <i>s1</i> (ANSI)                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
| <b>SYNOPSIS</b>      | <pre>size_t strxfrm (     char *      s1,          /* string out */     const char * s2,          /* string in */     size_t     n             /* size of buffer */ )</pre>                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| <b>DESCRIPTION</b>   | This routine transforms string <i>s2</i> and places the resulting string in <i>s1</i> . The transformation is such that if <b>strcmp()</b> is applied to two transformed strings, it returns a value greater than, equal to, or less than zero, corresponding to the result of the <b>strcoll()</b> function applied to the same two original strings. No more than <i>n</i> characters are placed into the resulting <i>s1</i> , including the terminating null character. If <i>n</i> is zero, <i>s1</i> is permitted to be a NULL pointer. If copying takes place between objects that overlap, the behavior is undefined. |
| <b>INCLUDE FILES</b> | <code>string.h</code>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| <b>RETURNS</b>       | The length of the transformed string, not including the terminating null character. If the value is <i>n</i> or more, the contents of <i>s1</i> are indeterminate.                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| <b>SEE ALSO</b>      | <code>ansiString</code> , <code>strcmp()</code> , <code>strcoll()</code>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |

---

## swab()

|                 |                                                                                                                                                                                                                 |
|-----------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>     | <code>swab()</code> – swap bytes                                                                                                                                                                                |
| <b>SYNOPSIS</b> | <pre>void swab (     char * source,          /* pointer to source buffer */     char * destination,    /* pointer to destination buffer */     int   nbytes           /* number of bytes to exchange */ )</pre> |

**DESCRIPTION** This routine gets the specified number of bytes from *source*, exchanges the adjacent even and odd bytes, and puts them in *destination*. The buffers *source* and *destination* should not overlap.

---

**NOTE:** On some CPUs, **swab()** will cause an exception if the buffers are unaligned. In such cases, use **uswab()** for unaligned swaps. On ARM family CPUs, **swab()** may reorder the bytes incorrectly without causing an exception if the buffers are unaligned. Again, use **uswab()** for unaligned swaps.

---

It is an error for *nbytes* to be odd.

**RETURNS** N/A

**SEE ALSO** **bLib**, **uswab()**

---

## symAdd()

**NAME** **symAdd()** – create and add a symbol to a symbol table, including a group number

**SYNOPSIS**

```
STATUS symAdd
(
    SYMTAB_ID symTblId,      /* symbol table to add symbol to */
    char *    name,         /* pointer to symbol name string */
    char *    value,        /* symbol address */
    SYM_TYPE  type,         /* symbol type */
    UINT16    group         /* symbol group */
)
```

**DESCRIPTION** This routine allocates a symbol *name* and adds it to a specified symbol table *symTblId* with the specified parameters *value*, *type*, and *group*. The *group* parameter specifies the group number assigned to a module when it is loaded; see the manual entry for **moduleLib**.

**RETURNS** OK, or ERROR if the symbol table is invalid or there is insufficient memory for the symbol to be allocated.

**SEE ALSO** **symLib**, **moduleLib**

---

## symByValueAndTypeFind()

**NAME** `symByValueAndTypeFind()` – look up a symbol by value and type

**SYNOPSIS**

```
STATUS symByValueAndTypeFind
(
    SYMTAB_ID  symTblId,      /* ID of symbol table to look in */
    UINT       value,        /* value of symbol to find */
    char * *   pName,        /* where to return symbol name string */
    int *      pValue,       /* where to put symbol value */
    SYM_TYPE * pType,        /* where to put symbol type */
    SYM_TYPE   sType,        /* symbol type to look for */
    SYM_TYPE   mask         /* bits in sType to pay attention to */
)
```

**DESCRIPTION** This routine searches a symbol table for a symbol matching both value and type (*value* and *sType*). If there is no matching entry, it chooses the table entry with the next lower value (among entries with the same type). A pointer to the symbol name string (with terminating EOS) is returned into *pName*. The actual value and the type are copied to *pValue* and *pType*. The *mask* parameter can be used to match sub-classes of type.

*pName* is a pointer to memory allocated by `symByValueAndTypeFind()`; the memory must be freed by the caller after the use of *pName*.

To search the global VxWorks symbol table, specify `sysSymTbl` as *symTblId*.

**RETURNS** `OK` or `ERROR` if *symTblId* is invalid, *pName* is `NULL`, or *value* is less than the lowest value in the table.

**SEE ALSO** `symLib`

## symByValueFind()

**NAME** symByValueFind() – look up a symbol by value

**SYNOPSIS**

```
STATUS symByValueFind
(
    SYMTAB_ID symTblId,      /* ID of symbol table to look in */
    UINT      value,        /* value of symbol to find */
    char * *  pName,        /* where return symbol name string */
    int *     pValue,       /* where to put symbol value */
    SYM_TYPE * pType        /* where to put symbol type */
)
```

**DESCRIPTION** This routine searches a symbol table for a symbol matching a specified value. If there is no matching entry, it chooses the table entry with the next lower value. A pointer to a copy of the symbol name string (with terminating EOS) is returned into *pName*. The actual value and the type are copied to *pValue* and *pType*.

*pName* is a pointer to memory allocated by `symByValueFind`; the memory must be freed by the caller after the use of *pName*.

To search the global VxWorks symbol table, specify `sysSymTbl` as *symTblId*.

**RETURNS** OK or ERROR if *symTblId* is invalid, *pName* is NULL, or *value* is less than the lowest value in the table.

**SEE ALSO** symLib

---

## symEach()

**NAME** symEach() – call a routine to examine each entry in a symbol table

**SYNOPSIS**

```
SYMBOL *symEach
(
    SYMTAB_ID symTblId,      /* pointer to symbol table */
    FUNCPTR   routine,      /* func to call for each tbl entry */
    int       routineArg    /* arbitrary user-supplied arg */
)
```

**DESCRIPTION** This routine calls a user-supplied routine to examine each entry in the symbol table; it calls the specified routine once for each entry. The routine should be declared as follows:

```

BOOL routine
(
    char    *name, /* entry name          */
    int     val,  /* value associated with entry */
    SYM_TYPE type, /* entry type                */
    int     arg,  /* arbitrary user-supplied arg */
    UINT16  group /* group number              */
)

```

The user-supplied routine should return **TRUE** if **symEach()** is to continue calling it for each entry, or **FALSE** if it is done and **symEach()** can exit.

**RETURNS** A pointer to the last symbol reached, or **NULL** if all symbols are reached.

**SEE ALSO** **symLib**

---

## symFindByName()

**NAME** **symFindByName()** – look up a symbol by name

**SYNOPSIS**

```

STATUS symFindByName
(
    SYMTAB_ID symTblId, /* ID of symbol table to look in */
    char *    name,     /* symbol name to look for */
    char *    *pValue,  /* where to put symbol value */
    SYM_TYPE * pType    /* where to put symbol type */
)

```

**DESCRIPTION** This routine searches a symbol table for a symbol matching a specified name. If the symbol is found, its value and type are copied to *pValue* and *pType*. If multiple symbols have the same name but differ in type, the routine chooses the matching symbol most recently added to the symbol table.

To search the global VxWorks symbol table, specify **sysSymTbl** as *symTblId*.

**RETURNS** **OK**, or **ERROR** if the symbol table ID is invalid or the symbol cannot be found.

**SEE ALSO** **symLib**

## symFindByNameAndType()

**NAME** symFindByNameAndType() – look up a symbol by name and type

**SYNOPSIS**

```
STATUS symFindByNameAndType
(
    SYMTAB_ID symTblId,      /* ID of symbol table to look in */
    char *    name,         /* symbol name to look for */
    char *    *pValue,      /* where to put symbol value */
    SYM_TYPE * pType,       /* where to put symbol type */
    SYM_TYPE  sType,        /* symbol type to look for */
    SYM_TYPE  mask          /* bits in sType to pay attention to */
)
```

**DESCRIPTION** This routine searches a symbol table for a symbol matching both name and type (*name* and *sType*). If the symbol is found, its value and type are copied to *pValue* and *pType*. The *mask* parameter can be used to match sub-classes of type.

To search the global VxWorks symbol table, specify **sysSymTbl** as *symTblId*.

**RETURNS** OK, or ERROR if the symbol table ID is invalid or the symbol is not found.

**SEE ALSO** symLib

---

## symFindByValue()

**NAME** symFindByValue() – look up a symbol by value

**SYNOPSIS**

```
STATUS symFindByValue
(
    SYMTAB_ID symTblId,      /* ID of symbol table to look in */
    UINT      value,        /* value of symbol to find */
    char *    name,         /* where to put symbol name string */
    int *     pValue,       /* where to put symbol value */
    SYM_TYPE * pType        /* where to put symbol type */
)
```

**DESCRIPTION** This routine is obsolete. It is replaced by **symByValueFind()**.

This routine searches a symbol table for a symbol matching a specified value. If there is no matching entry, it chooses the table entry with the next lower value. The symbol name (with terminating EOS), the actual value, and the type are copied to *name*, *pValue*, and *pType*.

For the *name* buffer, allocate `MAX_SYS_SYM_LEN + 1` bytes. The value `MAX_SYS_SYM_LEN` is defined in `sysSymTbl.h`. If the name of the symbol is longer than `MAX_SYS_SYM_LEN` bytes, it will be truncated to fit into the buffer. Whether or not the name was truncated, the string returned in the buffer will be null-terminated.

To search the global VxWorks symbol table, specify `sysSymTbl` as *symTblId*.

**RETURNS** OK, or **ERROR** if *symTblId* is invalid or *value* is less than the lowest value in the table.

**SEE ALSO** `symLib`

---

## symFindByValueAndType()

**NAME** `symFindByValueAndType()` – look up a symbol by value and type

**SYNOPSIS**

```

STATUS symFindByValueAndType
(
    SYMTAB_ID symTblId,      /* ID of symbol table to look in */
    UINT      value,        /* value of symbol to find */
    char *    name,         /* where to put symbol name string */
    int *     pValue,       /* where to put symbol value */
    SYM_TYPE * pType,       /* where to put symbol type */
    SYM_TYPE  sType,        /* symbol type to look for */
    SYM_TYPE  mask          /* bits in sType to pay attention to */
)
    
```

**DESCRIPTION** This routine is obsolete. It is replaced by the routine `symByValueAndTypeFind()`.

This routine searches a symbol table for a symbol matching both value and type (*value* and *sType*). If there is no matching entry, it chooses the table entry with the next lower value. The symbol name (with terminating EOS), the actual value, and the type are copied to *name*, *pValue*, and *pType*. The *mask* parameter can be used to match sub-classes of type.

For the *name* buffer, allocate `MAX_SYS_SYM_LEN + 1` bytes. The value `MAX_SYS_SYM_LEN` is defined in `sysSymTbl.h`. If the name of the symbol is longer than `MAX_SYS_SYM_LEN` bytes, it will be truncated to fit into the buffer. Whether or not the name was truncated, the string returned in the buffer will be null-terminated.

To search the global VxWorks symbol table, specify `sysSymTbl` as *symTblId*.

**RETURNS** OK, or ERROR if *symTblId* is invalid or *value* is less than the lowest value in the table. \*

**SEE ALSO** symLib

---

## symLibInit()

**NAME** symLibInit() – initialize the symbol table library

**SYNOPSIS** `STATUS symLibInit (void)`

**DESCRIPTION** This routine initializes the symbol table package. If the configuration macro `INCLUDE_SYM_TBL` is defined, `symLibInit()` is called by the root task, `usrRoot()`, in `usrConfig.c`.

**RETURNS** OK, or ERROR if the library could not be initialized.

**SEE ALSO** symLib

---

## symRemove()

**NAME** symRemove() – remove a symbol from a symbol table

**SYNOPSIS** `STATUS symRemove`

```
(  
    SYMTAB_ID symTblId,      /* symbol tbl to remove symbol from */  
    char *    name,          /* name of symbol to remove */  
    SYM_TYPE  type           /* type of symbol to remove */  
)
```

**DESCRIPTION** This routine removes a symbol of matching name and type from a specified symbol table. The symbol is deallocated if found. Note that VxWorks symbols in a standalone VxWorks image (where the symbol table is linked in) cannot be removed.

**RETURNS** OK, or ERROR if the symbol is not found or could not be deallocated.

**SEE ALSO** symLib

---

## symSyncLibInit()

**NAME** `symSyncLibInit()` – initialize host/target symbol table synchronization

**SYNOPSIS** `void symSyncLibInit ()`

**DESCRIPTION** This routine initializes host/target symbol table synchronization. To enable synchronization, it must be called before a target server is started. It is called automatically if the configuration macro `INCLUDE_SYM_TBL_SYNC` is defined.

**RETURNS** N/A

**SEE ALSO** `symSyncLib`

---

## symSyncTimeoutSet()

**NAME** `symSyncTimeoutSet()` – set WTX timeout

**SYNOPSIS** `UINT32 symSyncTimeoutSet`  
`(`  
`UINT32 timeout                   /* WTX timeout in milliseconds */`  
`)`

**DESCRIPTION** This routine sets the WTX timeout between target server and synchronization task.

**RETURNS** If *timeout* is 0, the current timeout, otherwise the new timeout value in milliseconds.

**SEE ALSO** `symSyncLib`

## symTblCreate()

**NAME** symTblCreate() – create a symbol table

**SYNOPSIS**

```
SYMTAB_ID symTblCreate
(
    int      hashSizeLog2,      /* size of hash table as a power of 2 */
    BOOL     sameNameOk,       /* allow 2 symbols of same name & type */
    PART_ID  symPartId         /* memory part ID for symbol allocation */
)
```

**DESCRIPTION** This routine creates and initializes a symbol table with a hash table of a specified size. The size of the hash table is specified as a power of two. For example, if *hashSizeLog2* is 6, a 64-entry hash table is created.

If *sameNameOk* is **FALSE**, attempting to add a symbol with the same name and type as an already-existing symbol results in an error.

Memory for storing symbols as they are added to the symbol table will be allocated from the memory partition *symPartId*. The ID of the system memory partition is stored in the global variable **memSysPartId**, which is declared in **memLib.h**.

**RETURNS** Symbol table ID, or NULL if memory is insufficient.

**SEE ALSO** symLib

---

## symTblDelete()

**NAME** symTblDelete() – delete a symbol table

**SYNOPSIS**

```
STATUS symTblDelete
(
    SYMTAB_ID symTblId         /* ID of symbol table to delete */
)
```

**DESCRIPTION** This routine deletes a specified symbol table. It deallocates all associated memory, including the hash table, and marks the table as invalid.

Deletion of a table that still contains symbols results in **ERROR**. Successful deletion includes the deletion of the internal hash table and the deallocation of memory associated with the table. The table is marked invalid to prohibit any future references.

**RETURNS** OK, or **ERROR** if the symbol table ID is invalid.

**SEE ALSO** `symLib`

---

## sysAuxClkConnect()

**NAME** `sysAuxClkConnect()` – connect a routine to the auxiliary clock interrupt

**SYNOPSIS** `STATUS sysAuxClkConnect`  
(  
    **FUNCPTR** routine, /\* routine called at each aux clock interrupt \*/  
    **int** arg /\* argument to auxiliary clock interrupt routine \*/  
)

**DESCRIPTION** This routine specifies the interrupt service routine to be called at each auxiliary clock interrupt. It does not enable auxiliary clock interrupts.

---

**NOTE:** This is a generic page for a BSP-specific routine; this description contains general information only. To determine if this call is supported by your BSP, or for information specific to your BSP's version of this routine, see the reference pages for your BSP.

---

**RETURNS** OK, or **ERROR** if the routine cannot be connected to the interrupt.

**SEE ALSO** `sysLib`, `intConnect()`, `sysAuxClkEnable()`, and BSP-specific reference pages for this routine.

---

## sysAuxClkDisable()

**NAME** `sysAuxClkDisable()` – turn off auxiliary clock interrupts

**SYNOPSIS** `void sysAuxClkDisable (void)`

**DESCRIPTION** This routine disables auxiliary clock interrupts.

---

**NOTE:** This is a generic page for a BSP-specific routine; this description contains general information only. To determine if this call is supported by your BSP, or for information specific to your BSP's version of this routine, see the reference pages for your BSP.

---

**RETURNS** N/A

**SEE ALSO** `sysLib`, `sysAuxClkEnable()`, and BSP-specific reference pages for this routine.

---

## sysAuxClkEnable()

**NAME** `sysAuxClkEnable()` – turn on auxiliary clock interrupts

**SYNOPSIS** `void sysAuxClkEnable (void)`

**DESCRIPTION** This routine enables auxiliary clock interrupts.

---

**NOTE:** This is a generic page for a BSP-specific routine; this description contains general information only. To determine if this call is supported by your BSP, or for information specific to your BSP's version of this routine, see the reference pages for your BSP.

---

**RETURNS** N/A

**SEE ALSO** `sysLib`, `sysAuxClkConnect()`, `sysAuxClkDisable()`, `sysAuxClkRateSet()`, and BSP-specific reference pages for this routine.

---

## sysAuxClkRateGet()

**NAME** `sysAuxClkRateGet()` – get the auxiliary clock rate

**SYNOPSIS** `int sysAuxClkRateGet (void)`

**DESCRIPTION** This routine returns the interrupt rate of the auxiliary clock.

---

**NOTE:** This is a generic page for a BSP-specific routine; this description contains general information only. To determine if this call is supported by your BSP, or for information specific to your BSP's version of this routine, see the reference pages for your BSP.

---

**RETURNS** The number of ticks per second of the auxiliary clock.

**SEE ALSO** `sysLib`, `sysAuxClkEnable()`, `sysAuxClkRateSet()`, and BSP-specific reference pages for this routine.

---

## sysAuxClkRateSet()

|                    |                                                                                                                                                                                                                                                                                                                                                                                                                        |
|--------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>        | <code>sysAuxClkRateSet()</code> – set the auxiliary clock rate                                                                                                                                                                                                                                                                                                                                                         |
| <b>SYNOPSIS</b>    | <pre><b>STATUS</b> sysAuxClkRateSet (     int ticksPerSecond    /* number of clock interrupts per second */ )</pre>                                                                                                                                                                                                                                                                                                    |
| <b>DESCRIPTION</b> | <p>This routine sets the interrupt rate of the auxiliary clock. It does not enable auxiliary clock interrupts.</p> <hr/> <p><b>NOTE:</b> This is a generic page for a BSP-specific routine; this description contains general information only. To determine if this call is supported by your BSP, or for information specific to your BSP's version of this routine, see the reference pages for your BSP.</p> <hr/> |
| <b>RETURNS</b>     | OK, or <b>ERROR</b> if the tick rate is invalid or the timer cannot be set.                                                                                                                                                                                                                                                                                                                                            |
| <b>SEE ALSO</b>    | <code>sysLib</code> , <code>sysAuxClkEnable()</code> , <code>sysAuxClkRateGet()</code> , and BSP-specific reference pages for this routine.                                                                                                                                                                                                                                                                            |

---

## sysBspRev()

|                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
|--------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>        | <code>sysBspRev()</code> – return the BSP version and revision number                                                                                                                                                                                                                                                                                                                                                                                                  |
| <b>SYNOPSIS</b>    | <pre>char * sysBspRev (void)</pre>                                                                                                                                                                                                                                                                                                                                                                                                                                     |
| <b>DESCRIPTION</b> | <p>This routine returns a pointer to a BSP version and revision number, for example, 1.0/1. <b>BSP_REV</b> is concatenated to <b>BSP_VERSION</b> and returned.</p> <hr/> <p><b>NOTE:</b> This is a generic page for a BSP-specific routine; this description contains general information only. To determine if this call is supported by your BSP, or for information specific to your BSP's version of this routine, see the reference pages for your BSP.</p> <hr/> |
| <b>RETURNS</b>     | A pointer to the BSP version/revision string.                                                                                                                                                                                                                                                                                                                                                                                                                          |
| <b>SEE ALSO</b>    | <code>sysLib</code> , and BSP-specific reference pages for this routine.                                                                                                                                                                                                                                                                                                                                                                                               |

---

## sysBusIntAck()

**NAME** sysBusIntAck() – acknowledge a bus interrupt

**SYNOPSIS**

```
int sysBusIntAck
(
    int intLevel          /* interrupt level to acknowledge */
)
```

**DESCRIPTION** This routine acknowledges a specified VME bus interrupt level.

---

**NOTE:** This is a generic page for a BSP-specific routine; this description contains general information only. To determine if this call is supported by your BSP, or for information specific to your BSP's version of this routine, see the reference pages for your BSP.

---

**RETURNS** NULL.

**SEE ALSO** sysLib, sysBusIntGen(), and BSP-specific reference pages for this routine.

---

## sysBusIntGen()

**NAME** sysBusIntGen() – generate a bus interrupt

**SYNOPSIS**

```
STATUS sysBusIntGen
(
    int intLevel,          /* bus interrupt level to generate */
    int vector            /* interrupt vector to generate (0-255) */
)
```

**DESCRIPTION** This routine generates a bus interrupt for a specified level with a specified vector.

---

**NOTE:** This is a generic page for a BSP-specific routine; this description contains general information only. To determine if this call is supported by your BSP, or for information specific to your BSP's version of this routine, see the reference pages for your BSP.

---

**RETURNS** OK, or ERROR if *intLevel* is out of range or the board cannot generate a bus interrupt.

**SEE ALSO** sysLib, sysBusIntAck(), and BSP-specific reference pages for this routine.

---

## sysBusTas()

**NAME** `sysBusTas()` – test and set a location across the bus

**SYNOPSIS**

```
BOOL sysBusTas
(
    char * adrs          /* address to be tested and set */
)
```

**DESCRIPTION** This routine performs a test-and-set instruction across the backplane.

---

**NOTE:** This is a generic page for a BSP-specific routine; this description contains general information only. To determine if this call is supported by your BSP, or for information specific to your BSP's version of this routine, see the reference pages for your BSP.

---

**NOTE:** This routine is equivalent to `vxTas()`.

---

**RETURNS** TRUE if the value had not been set but is now, or FALSE if the value was set already.

**SEE ALSO** `sysLib`, `vxTas()`, and BSP-specific reference pages for this routine.

---

## sysBusToLocalAdrs()

**NAME** `sysBusToLocalAdrs()` – convert a bus address to a local address

**SYNOPSIS**

```
STATUS sysBusToLocalAdrs
(
    int   adrsSpace,      /* bus address space in which busAdrs resides */
    char * busAdrs,      /* bus address to convert */
    char * *pLocalAdrs   /* where to return local address */
)
```

**DESCRIPTION** This routine gets the local address that accesses a specified bus memory address.

---

**NOTE:** This is a generic page for a BSP-specific routine; this description contains general information only. To determine if this call is supported by your BSP, or for information specific to your BSP's version of this routine, see the reference pages for your BSP.

---

**RETURNS** OK, or ERROR if the address space is unknown or the mapping is not possible.

**SEE ALSO** `sysLib`, `sysLocalToBusAdrs()`, and BSP-specific reference pages for this routine.

---

## sysClkConnect()

**NAME** `sysClkConnect()` – connect a routine to the system clock interrupt

**SYNOPSIS**

```
STATUS sysClkConnect
(
    FUNCPTR routine,          /* routine called at each system clock */
                                /* interrupt */
    int    arg                /* argument with which to call routine */
)
```

**DESCRIPTION** This routine specifies the interrupt service routine to be called at each clock interrupt. Normally, it is called from `usrRoot()` in `usrConfig.c` to connect `usrClock()` to the system clock interrupt.

---

**NOTE:** This is a generic page for a BSP-specific routine; this description contains general information only. To determine if this call is supported by your BSP, or for information specific to your BSP's version of this routine, see the reference entries for your BSP.

---

**RETURN** OK, or ERROR if the routine cannot be connected to the interrupt.

**SEE ALSO** `sysLib`, `intConnect()`, `usrClock()`, `sysClkEnable()`, and BSP-specific reference pages for this routine.

---

## sysClkDisable()

**NAME** `sysClkDisable()` – turn off system clock interrupts

**SYNOPSIS**

```
void sysClkDisable (void)
```

**DESCRIPTION** This routine disables system clock interrupts.

---

**NOTE:** This is a generic page for a BSP-specific routine; this description contains general information only. To determine if this call is supported by your BSP, or for information specific to your BSP's version of this routine, see the reference pages for your BSP.

---

**RETURNS** N/A

**SEE ALSO** `sysLib`, `sysClkEnable()`, and BSP-specific reference pages for this routine.

---

## sysClkEnable()

**NAME** `sysClkEnable()` – turn on system clock interrupts

**SYNOPSIS** `void sysClkEnable (void)`

**DESCRIPTION** This routine enables system clock interrupts.

---

**NOTE:** This is a generic page for a BSP-specific routine; this description contains general information only. To determine if this call is supported by your BSP, or for information specific to your BSP's version of this routine, see the reference pages for your BSP.

---

**RETURNS** N/A

**SEE ALSO** `sysLib`, `sysClkConnect()`, `sysClkDisable()`, `sysClkRateSet()`, and BSP-specific reference pages for this routine.

---

## sysClkRateGet()

**NAME** `sysClkRateGet()` – get the system clock rate

**SYNOPSIS** `int sysClkRateGet (void)`

**DESCRIPTION** This routine returns the system clock rate.

---

**NOTE:** This is a generic page for a BSP-specific routine; this description contains general information only. To determine if this call is supported by your BSP, or for information specific to your BSP's version of this routine, see the reference pages for your BSP.

---

**RETURNS** The number of ticks per second of the system clock.

**SEE ALSO** `sysLib`, `sysClkEnable()`, `sysClkRateSet()`, and BSP-specific reference pages for this routine.

---

## sysClkRateSet()

**NAME** sysClkRateSet() – set the system clock rate

**SYNOPSIS**

```
STATUS sysClkRateSet
(
    int ticksPerSecond    /* number of clock interrupts per second */
)
```

**DESCRIPTION** This routine sets the interrupt rate of the system clock. It is called by **usrRoot()** in **usrConfig.c**.

There may be interactions between this routine and the POSIX **clockLib** routines. Refer to the **clockLib** reference entry.

---

**NOTE:** This is a generic page for a BSP-specific routine; this description contains general information only. To determine if this call is supported by your BSP, or for information specific to your BSP's version of this routine, see the reference pages for your BSP.

---

**RETURNS** OK, or **ERROR** if the tick rate is invalid or the timer cannot be set.

**SEE ALSO** **sysLib**, **sysClkEnable()**, **sysClkRateGet()**, **clockLib**, and BSP-specific reference pages for this routine.

---

## sysHwInit()

**NAME** sysHwInit() – initialize the system hardware

**SYNOPSIS**

```
void sysHwInit (void)
```

**DESCRIPTION** This routine initializes various features of the board. It is called from **usrInit()** in **usrConfig.c**.

---

**NOTE:** This is a generic page for a BSP-specific routine; this description contains general information only. To determine if this call is supported by your BSP, or for information specific to your BSP's version of this routine, see the reference pages for your BSP.

---

**NOTE:** This routine should not be called directly by the user application.

---

**RETURNS** N/A

**SEE ALSO** `sysLib`, and BSP-specific reference pages for this routine.

---

## sysIntDisable()

**NAME** `sysIntDisable()` – disable a bus interrupt level

**SYNOPSIS**

```
STATUS sysIntDisable
(
    int intLevel          /* interrupt level to disable */
)
```

**DESCRIPTION** This routine disables a specified bus interrupt level.

---

**NOTE:** This is a generic page for a BSP-specific routine; this description contains general information only. To determine if this call is supported by your BSP, or for information specific to your BSP's version of this routine, see the reference pages for your BSP.

---

**RETURNS** OK, or ERROR if *intLevel* is out of range.

**SEE ALSO** `sysLib`, `sysIntEnable()`, and BSP-specific reference pages for this routine.

---

## sysIntEnable()

**NAME** `sysIntEnable()` – enable a bus interrupt level

**SYNOPSIS**

```
STATUS sysIntEnable
(
    int intLevel          /* interrupt level to enable (1-7) */
)
```

**DESCRIPTION** This routine enables a specified bus interrupt level.

---

**NOTE:** This is a generic page for a BSP-specific routine; this description contains general information only. To determine if this call is supported by your BSP, or for information specific to your BSP's version of this routine, see the reference pages for your BSP.

---

**RETURNS** OK, or ERROR if *intLevel* is out of range.

**SEE ALSO** `sysLib`, `sysIntDisable()`, and BSP-specific reference pages for this routine.

---

## sysLocalToBusAdrs()

**NAME** `sysLocalToBusAdrs()` – convert a local address to a bus address

**SYNOPSIS**

```
STATUS sysLocalToBusAdrs
(
    int    adrsSpace,      /* bus address space in which busAdrs resides */
    char * localAdrs,     /* local address to convert */
    char * *pBusAdrs      /* where to return bus address */
)
```

**DESCRIPTION** This routine gets the bus address that accesses a specified local memory address.

---

**NOTE:** This is a generic page for a BSP-specific routine; this description contains general information only. To determine if this call is supported by your BSP, or for information specific to your BSP's version of this routine, see the reference pages for your BSP.

---

**RETURNS** OK, or ERROR if the address space is unknown or not mapped.

**SEE ALSO** `sysLib`, `sysBusToLocalAdrs()`, and BSP-specific reference pages for this routine.

---

## sysMailboxConnect()

**NAME** `sysMailboxConnect()` – connect a routine to the mailbox interrupt

**SYNOPSIS**

```
STATUS sysMailboxConnect
(
    FUNCPTR routine,      /* routine called at each mailbox interrupt */
    int    arg            /* argument with which to call routine */
)
```

**DESCRIPTION** This routine specifies the interrupt service routine to be called at each mailbox interrupt.

---

**NOTE:** This is a generic page for a BSP-specific routine; this description contains general information only. To determine if this call is supported by your BSP, or for information specific to your BSP's version of this routine, see the reference pages for your BSP.

---

**RETURNS** OK, or **ERROR** if the routine cannot be connected to the interrupt.

**SEE ALSO** `sysLib`, `intConnect()`, `sysMailboxEnable()`, and BSP-specific reference pages for this routine.

---

## sysMailboxEnable()

**NAME** `sysMailboxEnable()` – enable the mailbox interrupt

**SYNOPSIS**

```
STATUS sysMailboxEnable  
(  
    char * mailboxAdrs        /* address of mailbox (ignored) */  
)
```

**DESCRIPTION** This routine enables the mailbox interrupt.

---

**NOTE:** This is a generic page for a BSP-specific routine; this description contains general information only. To determine if this call is supported by your BSP, or for information specific to your BSP's version of this routine, see the reference pages for your BSP.

---

**RETURNS** OK, always.

**SEE ALSO** `sysLib`, `sysMailboxConnect()`, and BSP-specific reference pages for this routine.

---

## sysMemTop()

**NAME** `sysMemTop()` – get the address of the top of logical memory

**SYNOPSIS**

```
char *sysMemTop (void)
```

**DESCRIPTION** This routine returns the address of the top of memory.

---

**NOTE:** This is a generic page for a BSP-specific routine; this description contains general information only. To determine if this call is supported by your BSP, or for information specific to your BSP's version of this routine, see the reference pages for your BSP.

---

**RETURNS** The address of the top of memory.

**SEE ALSO** `sysLib`, and BSP-specific reference pages for this routine.

---

## sysModel()

**NAME** `sysModel()` – return the model name of the CPU board

**SYNOPSIS** `char *sysModel (void)`

**DESCRIPTION** This routine returns the model name of the CPU board.

---

**NOTE:** This is a generic page for a BSP-specific routine; this description contains general information only. To determine if this call is supported by your BSP, or for information specific to your BSP's version of this routine, see the reference pages for your BSP.

---

**RETURNS** A pointer to a string containing the board name.

**SEE ALSO** `sysLib`, and BSP-specific reference pages for this routine.

---

## sysNanoDelay()

**NAME** `sysNanoDelay()` – delay for specified number of nanoseconds

**SYNOPSIS**

```
void sysNanoDelay
(
    UINT32 nanoseconds    /* nanoseconds to delay */
)
```

**DESCRIPTION** This is an optional API for BSPs to provide. Some, but not all, drivers do require the BSP to implement this function.

When implemented, this function implements a spin loop type delay for at least the specified number of nanoseconds. This is not a task delay, control of the processor is not given up to another task. The actual delay must be equal to or greater than the requested number of nanoseconds.

The purpose of this function is to provide a reasonably accurate time delay of very short duration. It should not be used for any delays that are much greater than two system clock ticks in length. For delays of a full clock tick, or more, the use of **taskDelay( )** is recommended.

This routine should be implemented as interrupt safe.

---

**NOTE:** This is a generic page for a BSP-specific routine; this description contains general information only. To determine if this call is supported by your BSP, or for information specific to your BSP's version of this routine, see the reference pages for your BSP.

---

**RETURNS** N/A.

**SEE ALSO** **sysLib**, and BSP-specific reference pages for this routine.

---

## sysNvRamGet( )

**NAME** **sysNvRamGet( )** – get the contents of non-volatile RAM

**SYNOPSIS**

```
STATUS sysNvRamGet  
(  
    char * string,          /* where to copy non-volatile RAM */  
    int  strLen,           /* maximum number of bytes to copy */  
    int  offset            /* byte offset into non-volatile RAM */  
)
```

**DESCRIPTION** This routine copies the contents of non-volatile memory into a specified string. The string will be terminated with an EOS.

---

**NOTE:** This is a generic page for a BSP-specific routine; this description contains general information only. To determine if this call is supported by your BSP, or for information specific to your BSP's version of this routine, see the reference pages for your BSP.

---

**RETURNS** OK, or **ERROR** if access is outside the non-volatile RAM address range.

**SEE ALSO** **sysLib**, **sysNvRamSet( )**, and BSP-specific reference pages for this routine.

---

## sysNvRamSet()

**NAME** sysNvRamSet() – write to non-volatile RAM

**SYNOPSIS**

```
STATUS sysNvRamSet
(
    char * string,          /* string to be copied into non-volatile RAM */
    int   strLen,          /* maximum number of bytes to copy */
    int   offset           /* byte offset into non-volatile RAM */
)
```

**DESCRIPTION** This routine copies a specified string into non-volatile RAM.

---

**NOTE:** This is a generic page for a BSP-specific routine; this description contains general information only. To determine if this call is supported by your BSP, or for information specific to your BSP's version of this routine, see the reference pages for your BSP.

---

**RETURNS** OK, or ERROR if access is outside the non-volatile RAM address range.

**SEE ALSO** sysLib, sysNvRamGet(), and BSP-specific reference pages for this routine.

---

## sysPhysMemTop()

**NAME** sysPhysMemTop() – get the address of the top of memory

**SYNOPSIS** char \* sysPhysMemTop (void)

**DESCRIPTION** This routine returns the address of the first missing byte of memory, which indicates the top of memory.

Normally, the amount of physical memory is specified with the macro LOCAL\_MEM\_SIZE. BSPs that support run-time memory sizing do so only if the macro LOCAL\_MEM\_AUTOSIZE is defined. If not defined, then LOCAL\_MEM\_SIZE is assumed to be, and must be, the true size of physical memory.

---

**NOTE:** Do not adjust LOCAL\_MEM\_SIZE to reserve memory for application use. See sysMemTop() for more information on reserving memory.

---

---

**NOTE:** This is a generic page for a BSP-specific routine; this description contains general information only. To determine if this call is supported by your BSP, or for information specific to your BSP's version of this routine, see the reference pages for your BSP.

---

**RETURNS** The address of the top of physical memory.

**SEE ALSO** `sysLib`, `sysMemTop()`, and BSP-specific reference pages for this routine.

---

## sysProcNumGet()

**NAME** `sysProcNumGet()` – get the processor number

**SYNOPSIS** `int sysProcNumGet (void)`

**DESCRIPTION** This routine returns the processor number for the CPU board, which is set with `sysProcNumSet()`.

---

**NOTE:** This is a generic page for a BSP-specific routine; this description contains general information only. To determine if this call is supported by your BSP, or for information specific to your BSP's version of this routine, see the reference pages for your BSP.

---

**RETURNS** The processor number for the CPU board.

**SEE ALSO** `sysLib`, `sysProcNumSet()`, and BSP-specific reference pages for this routine.

---

## sysProcNumSet()

**NAME** `sysProcNumSet()` – set the processor number

**SYNOPSIS** `void sysProcNumSet`  
`(`  
`int procNum                   /* processor number */`  
`)`

**DESCRIPTION** This routine sets the processor number for the CPU board. Processor numbers should be unique on a single backplane.

---

**NOTE:** This is a generic page for a BSP-specific routine; this description contains general information only. To determine if this call is supported by your BSP, or for information specific to your BSP's version of this routine, see the reference pages for your BSP.

---

**RETURNS** N/A

**SEE ALSO** `sysLib`, `sysProcNumGet()`, and BSP-specific reference pages for this routine.

---

## sysScsiBusReset()

**NAME** `sysScsiBusReset()` – assert the RST line on the SCSI bus (Western Digital WD33C93 only)

**SYNOPSIS**

```
void sysScsiBusReset
(
    WD_33C93_SCSI_CTRL * pSbic /* ptr to SBIC info */
)
```

**DESCRIPTION** This routine asserts the RST line on the SCSI bus, which causes all connected devices to return to a quiescent state.

---

**NOTE:** This is a generic page for a BSP-specific routine; this description contains general information only. To determine if this call is supported by your BSP, or for information specific to your BSP's version of this routine, see the reference pages for your BSP.

---

**RETURNS** N/A

**SEE ALSO** `sysLib`, and BSP-specific reference pages for this routine.

---

## sysScsiConfig()

**NAME** `sysScsiConfig()` – system SCSI configuration

**SYNOPSIS** `STATUS sysScsiConfig (void)`

**DESCRIPTION** This is an example SCSI configuration routine.

Most of the code found here is an example of how to declare a SCSI peripheral configuration. You must edit this routine to reflect the actual configuration of your SCSI bus. This example can also be found in `src/config/usrScsi.c`.

If you are just getting started, you can test your hardware configuration by defining `SCSI_AUTO_CONFIG`, which will probe the bus and display all devices found. No device should have the same SCSI bus ID as your VxWorks SCSI port (default = 7), or the same as any other device. Check for proper bus termination.

There are three configuration examples here. They demonstrate configuration of a SCSI hard disk (any type), an OMTI 3500 floppy disk, and a tape drive (any type).

**Hard Disk** The hard disk is divided into two 32-Mbyte partitions and a third partition with the remainder of the disk. The first partition is initialized as a dosFs device. The second and third partitions are initialized as rt11Fs devices, each with 256 directory entries.

It is recommended that the first partition (`BLK_DEV`) on a block device be a dosFs device, if the intention is eventually to boot VxWorks from the device. This will simplify the task considerably.

**Floppy Disk** The floppy, since it is a removable medium device, is allowed to have only a single partition, and dosFs is the file system of choice for this device, since it facilitates media compatibility with IBM PC machines.

In contrast to the hard disk configuration, the floppy setup in this example is more intricate. Note that the `scsiPhysDevCreate()` call is issued twice. The first time is merely to get a “handle” to pass to `scsiModeSelect()`, since the default media type is sometimes inappropriate (in the case of generic SCSI-to-floppy cards). After the hardware is correctly configured, the handle is discarded via `scsiPhysDevDelete()`, after which the peripheral is correctly configured by a second call to `scsiPhysDevCreate()`. (Before the `scsiModeSelect()` call, the configuration information was incorrect.) Note that after the `scsiBlkDevCreate()` call, the correct values for `sectorsPerTrack` and `nHeads` must be set via `scsiBlkDevInit()`. This is necessary for IBM PC compatibility.

**Tape Drive** The tape configuration is also somewhat complex because certain device parameters need to be turned off within VxWorks and the fixed-block size needs to be defined, assuming that the tape supports fixed blocks.

The last parameter to the **dosFsDevInit()** call is a pointer to a **DOS\_VOL\_CONFIG** structure. By specifying **NULL**, you are asking **dosFsDevInit()** to read this information off the disk in the drive. This may fail if no disk is present or if the disk has no valid **dosFs** directory. Should this be the case, you can use the **dosFsMkfs()** command to create a new directory on a disk. This routine uses default parameters (see **dosFsLib**) that may not be suitable for your application, in which case you should use **dosFsDevInit()** with a pointer to a valid **DOS\_VOL\_CONFIG** structure that you have created and initialized. If **dosFsDevInit()** is used, a **diskInit()** call should be made to write a new directory on the disk, if the disk is blank or disposable.

**NOTE** The variable **pSbdFloppy** is global to allow the above calls to be made from the VxWorks shell, for example:

```
-> dosFsMkfs "/fd0/", pSbdFloppy
```

If a disk is new, use **diskFormat()** to format it.

---

**NOTE:** This is a generic page for a BSP-specific routine; this description contains general information only. To determine if this call is supported by your BSP, or for information specific to your BSP's version of this routine, see the reference pages for your BSP.

---

**RETURNS** OK or ERROR.

**SEE ALSO** **sysLib**, and BSP-specific reference pages for this routine.

---

## sysScsiInit()

**NAME** **sysScsiInit()** – initialize an on-board SCSI port

**SYNOPSIS** **STATUS sysScsiInit (void)**

**DESCRIPTION** This routine creates and initializes a SCSI control structure, enabling use of the on-board SCSI port. It also connects the proper interrupt service routine to the desired vector, and enables the interrupt at the desired level.

If SCSI DMA is supported by the board and **INCLUDE\_SCSI\_DMA** is defined, the DMA is also initialized.

---

**NOTE:** This is a generic page for a BSP-specific routine; this description contains general information only. To determine if this call is supported by your BSP, or for information specific to your BSP's version of this routine, see the reference pages for your BSP.

---

**RETURNS** OK, or **ERROR** if the control structure cannot be connected, the controller cannot be initialized, or the DMA's interrupt cannot be connected.

**SEE ALSO** `sysLib`, and BSP-specific reference pages for this routine.

---

## sysSerialChanGet()

**NAME** `sysSerialChanGet()` – get the `SIO_CHAN` device associated with a serial channel

**SYNOPSIS**

```
SIO_CHAN * sysSerialChanGet
(
    int channel          /* serial channel */
)
```

**DESCRIPTION** This routine gets the `SIO_CHAN` device associated with a specified serial channel.

---

**NOTE:** This is a generic page for a BSP-specific routine; this description contains general information only. To determine if this call is supported by your BSP, or for information specific to your BSP's version of this routine, see the reference pages for your BSP.

---

**RETURNS** A pointer to the `SIO_CHAN` structure for the channel, or **ERROR** if the channel is invalid.

**SEE ALSO** `sysLib`, and BSP-specific reference pages for this routine.

---

## sysSerialHwInit()

**NAME** `sysSerialHwInit()` – initialize the BSP serial devices to a quiescent state

**SYNOPSIS** `void sysSerialHwInit (void)`

**DESCRIPTION** This routine initializes the BSP serial device descriptors and puts the devices in a quiescent state. It is called from `sysHwInit()` with interrupts locked.

---

**NOTE:** This is a generic page for a BSP-specific routine; this description contains general information only. To determine if this call is supported by your BSP, or for information specific to your BSP's version of this routine, see the reference pages for your BSP.

---

**RETURNS** N/A

**SEE ALSO** `sysLib`, and BSP-specific reference pages for this routine.

---

## sysSerialHwInit2()

**NAME** `sysSerialHwInit2()` – connect BSP serial device interrupts

**SYNOPSIS** `void sysSerialHwInit2 (void)`

**DESCRIPTION** This routine connects the BSP serial device interrupts. It is called from `sysHwInit2()`. Serial device interrupts could not be connected in `sysSerialHwInit()` because the kernel memory allocator was not initialized at that point, and `intConnect()` calls `malloc()`.

---

**NOTE:** This is a generic page for a BSP-specific routine; this description contains general information only. To determine if this call is supported by your BSP, or for information specific to your BSP's version of this routine, see the reference pages for your BSP.

---

**RETURNS** N/A

**SEE ALSO** `sysLib`, and BSP-specific reference pages for this routine.

---

## sysSerialReset()

**NAME** `sysSerialReset()` – reset all SIO devices to a quiet state

**SYNOPSIS** `void sysSerialReset (void)`

**DESCRIPTION** This routine is called from `sysToMonitor()` to reset all SIO device and prevent them from generating interrupts or performing DMA cycles.

---

**NOTE:** This is a generic page for a BSP-specific routine; this description contains general information only. To determine if this call is supported by your BSP, or for information specific to your BSP's version of this routine, see the reference pages for your BSP.

---

**RETURNS** N/A

**SEE ALSO** `sysLib`, and BSP-specific reference pages for this routine.

---

## system()

|                      |                                                                                |
|----------------------|--------------------------------------------------------------------------------|
| <b>NAME</b>          | <b>system()</b> – pass a string to a command processor (Unimplemented) (ANSI)  |
| <b>SYNOPSIS</b>      | <pre>int system (     const char * string      /* pointer to string */ )</pre> |
| <b>DESCRIPTION</b>   | This function is not applicable to VxWorks.                                    |
| <b>INCLUDE FILES</b> | stdlib.h                                                                       |
| <b>RETURNS</b>       | OK, always.                                                                    |
| <b>SEE ALSO</b>      | ansiStdlib                                                                     |

---

## sysToMonitor()

|                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
|--------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>        | <b>sysToMonitor()</b> – transfer control to the ROM monitor                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
| <b>SYNOPSIS</b>    | <pre>STATUS sysToMonitor (     int startType           /* parameter passed to ROM to tell it how */                            /* to boot */ )</pre>                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| <b>DESCRIPTION</b> | <p>This routine transfers control to the ROM monitor. Normally, it is called only by <b>reboot()</b>—which services CTRL+X—and by bus errors at interrupt level. However, in some circumstances, the user may wish to introduce a <i>startType</i> to enable special boot ROM facilities.</p> <hr/> <p><b>NOTE:</b> This is a generic page for a BSP-specific routine; this description contains general information only. To determine if this call is supported by your BSP, or for information specific to your BSP's version of this routine, see the reference pages for your BSP.</p> <hr/> |
| <b>RETURNS</b>     | Does not return.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
| <b>SEE ALSO</b>    | sysLib, and BSP-specific reference pages for this routine.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |

## **tan()**

|                      |                                                                                                      |
|----------------------|------------------------------------------------------------------------------------------------------|
| <b>NAME</b>          | <b>tan()</b> – compute a tangent (ANSI)                                                              |
| <b>SYNOPSIS</b>      | <pre>double tan (     double x          /* angle in radians */ )</pre>                               |
| <b>DESCRIPTION</b>   | This routine computes the tangent of $x$ in double precision. The angle $x$ is expressed in radians. |
| <b>INCLUDE FILES</b> | <b>math.h</b>                                                                                        |
| <b>RETURNS</b>       | The double-precision tangent of $x$ .                                                                |
| <b>SEE ALSO</b>      | <b>ansiMath, mathALib</b>                                                                            |

---

## **tanf()**

|                      |                                                                                                     |
|----------------------|-----------------------------------------------------------------------------------------------------|
| <b>NAME</b>          | <b>tanf()</b> – compute a tangent (ANSI)                                                            |
| <b>SYNOPSIS</b>      | <pre>float tanf (     float x          /* angle in radians */ )</pre>                               |
| <b>DESCRIPTION</b>   | This routine returns the tangent of $x$ in single precision. The angle $x$ is expressed in radians. |
| <b>INCLUDE FILES</b> | <b>math.h</b>                                                                                       |
| <b>RETURNS</b>       | The single-precision tangent of $x$ .                                                               |
| <b>SEE ALSO</b>      | <b>mathALib</b>                                                                                     |

---

## **tanh()**

|                      |                                                                                                                 |
|----------------------|-----------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>          | <b>tanh()</b> – compute a hyperbolic tangent (ANSI)                                                             |
| <b>SYNOPSIS</b>      | <pre>double tanh (     double x          /* number whose hyperbolic tangent is required */ )</pre>              |
| <b>DESCRIPTION</b>   | This routine returns the hyperbolic tangent of $x$ in double precision (IEEE double, 53 bits).                  |
| <b>INCLUDE FILES</b> | <b>math.h</b>                                                                                                   |
| <b>RETURNS</b>       | The double-precision hyperbolic tangent of $x$ .<br>Special cases:<br>If $x$ is NaN, <b>tanh()</b> returns NaN. |
| <b>SEE ALSO</b>      | <b>ansiMath, mathALib</b>                                                                                       |

---

## **tanhf()**

|                      |                                                                                                   |
|----------------------|---------------------------------------------------------------------------------------------------|
| <b>NAME</b>          | <b>tanhf()</b> – compute a hyperbolic tangent (ANSI)                                              |
| <b>SYNOPSIS</b>      | <pre>float tanhf (     float x          /* number whose hyperbolic tangent is required */ )</pre> |
| <b>DESCRIPTION</b>   | This routine returns the hyperbolic tangent of $x$ in single precision.                           |
| <b>INCLUDE FILES</b> | <b>math.h</b>                                                                                     |
| <b>RETURNS</b>       | The single-precision hyperbolic tangent of $x$ .                                                  |
| <b>SEE ALSO</b>      | <b>mathALib</b>                                                                                   |

---

## tapeFsDevInit()

**NAME** `tapeFsDevInit()` – associate a sequential device with tape volume functions

**SYNOPSIS**

```
TAPE_VOL_DESC *tapeFsDevInit
(
    char *      volName,      /* volume name */
    SEQ_DEV *   pSeqDev,      /* pointer to sequential device info */
    TAPE_CONFIG * pTapeConfig /* pointer to tape config info */
)
```

**DESCRIPTION**

This routine takes a sequential device created by a device driver and defines it as a tape file system volume. As a result, when high-level I/O operations, such as `open()` and `write()`, are performed on the device, the calls will be routed through `tapeFsLib`.

This routine associates `volName` with a device and installs it in the VxWorks I/O system-device table. The driver number used when the device is added to the table is that which was assigned to the tape library during `tapeFsInit()`. (The driver number is kept in the global variable `tapeFsDrvNum`.)

The `SEQ_DEV` structure specified by `pSeqDev` contains configuration data describing the device and the addresses of the routines which are called to read blocks, write blocks, write file marks, reset the device, check device status, perform other I/O control functions (`ioctl()`), reserve and release devices, load and unload devices, and rewind devices. These routines are not called until they are required by subsequent I/O operations. The `TAPE_CONFIG` structure is used to define configuration parameters for the `TAPE_VOL_DESC`. The configuration parameters are defined and described in `tapeFsLib.h`.

**RETURNS** A pointer to the volume descriptor (`TAPE_VOL_DESC`), or `NULL` if there is an error.

**ERRNO** `S_tapeFsLib_NO_SEQ_DEV`, `S_tapeFsLib_ILLEGAL_TAPE_CONFIG_PARM`

**SEE ALSO** `tapeFsLib`

---

## tapeFsInit()

- NAME** `tapeFsInit()` – initialize the tape volume library
- SYNOPSIS** `STATUS tapeFsInit ()`
- DESCRIPTION** This routine initializes the tape volume library. It must be called exactly once, before any other routine in the library. Only one file descriptor per volume is assumed.
- This routine also installs tape volume library routines in the VxWorks I/O system driver table. The driver number assigned to `tapeFsLib` is placed in the global variable `tapeFsDrvNum`. This number is later associated with system file descriptors opened to tapeFs devices.
- To enable this initialization, simply call the routine `tapeFsDevInit()`, which automatically calls `tapeFsInit()` in order to initialize the tape file system.
- RETURNS** OK or ERROR.
- SEE ALSO** `tapeFsLib`

---

## tapeFsReadyChange()

- NAME** `tapeFsReadyChange()` – notify `tapeFsLib` of a change in ready status
- SYNOPSIS** `STATUS tapeFsReadyChange`  
`(`  
`TAPE_VOL_DESC * pTapeVol /* pointer to volume descriptor */`  
`)`
- DESCRIPTION** This routine sets the volume descriptor state to `TAPE_VD_READY_CHANGED`. It should be called whenever a driver senses that a device has come on-line or gone off-line (for example, that a tape has been inserted or removed).
- After this routine has been called, the next attempt to use the volume results in an attempted remount.
- RETURNS** OK if the read change status is set, or ERROR if the file descriptor is in use.
- ERRNO** `S_tapeFsLib_FILE_DESCRIPTOR_BUSY`
- SEE ALSO** `tapeFsLib`

## tapeFsVolUnmount()

**NAME** `tapeFsVolUnmount()` – disable a tape device volume

**SYNOPSIS**

```
STATUS tapeFsVolUnmount
(
    TAPE_VOL_DESC * pTapeVol /* pointer to volume descriptor */
)
```

**DESCRIPTION** This routine is called when I/O operations on a volume are to be discontinued. This is commonly done before changing removable tape. All buffered data for the volume is written to the device (if possible), any open file descriptors are marked obsolete, and the volume is marked not mounted.

Because this routine flushes data from memory to the physical device, it should not be used in situations where the tape-change is not recognized until after a new tape has been inserted. In these circumstances, use the ready-change mechanism. (See the manual entry for `tapeFsReadyChange()`.)

This routine may also be called by issuing an `ioctl()` call using the `FIOUNMOUNT` function code.

**RETURNS** OK, or ERROR if the routine cannot access the volume.

**ERRNO** `S_tapeFsLib_VOLUME_NOT_AVAILABLE`, `S_tapeFsLib_FILE_DESCRIPTOR_BUSY`,  
`S_tapeFsLib_SERVICE_NOT_AVAILABLE`

**SEE ALSO** `tapeFsLib`, `tapeFsReadyChange()`

---

## tarArchive()

**NAME** `tarArchive()` – archive named file/dir onto tape in tar format

**SYNOPSIS**

```
STATUS tarArchive
(
    char * pTape,           /* tape device name */
    int   bfactor,         /* requested blocking factor */
    BOOL  verbose,         /* if TRUE print progress info */
    char * pName            /* file/dir name to archive */
)
```

**DESCRIPTION** This function creates a UNIX compatible tar formatted archives which contain entire file hierarchies from disk file systems. Files and directories are archived with mode and time information as returned by **stat()**.

The *tape* argument can be any tape drive device name or a name of any file that will be created if necessary, and will contain the archive. If *tape* is set to "-", standard output will be used. If *tape* is **NULL** (unspecified from Shell), the default archive file name stored in global variable *TAPE* will be used.

Each **write()** of the archive file will be exactly *bfactor*\*512 bytes long, hence on tapes in variable mode, this will be the physical block size on the tape. With Fixed Mode tapes this is only a performance matter. If *bfactor* is 0, or unspecified from Shell, it will be set to the default value of 20.

The *verbose* argument is a boolean, if set to 1, will cause informative messages to be printed to standard error whenever an action is taken, otherwise, only errors are reported.

The *name* argument is the path of the hierarchy to be archived. if **NULL** (or unspecified from the Shell), the current directory path "." will be used. This is the path as seen from the target, not from the Tornado host.

All informative and error message are printed to standard error.

---

**NOTE:** Refrain from specifying absolute path names in *path*, such archives tend to be either difficult to extract or can cause unexpected damage to existing files if such exist under the same absolute name.

---

There is no way of specifying a number of hierarchies to dump.

**SEE ALSO** tarLib

---

## tarExtract()

**NAME** tarExtract() – extract all files from a tar formatted tape

**SYNOPSIS**

```
STATUS tarExtract
(
  char * pTape,           /* tape device name */
  int   bfactor,         /* requested blocking factor */
  BOOL  verbose          /* if TRUE print progress info */
)
```

**DESCRIPTION** This is a UNIX-tar compatible utility that extracts entire file hierarchies from tar-formatted archive. The files are extracted with their original names and modes. In some cases a file

**tarToc()**

cannot be created on disk, for example if the name is too long for regular DOS file name conventions, in such cases entire files are skipped, and this program will continue with the next file. Directories are created in order to be able to create all files on tape.

The *tape* argument may be any tape device name or file name that contains a tar formatted archive. If *tape* is equal "-", standard input is used. If *tape* is NULL (or unspecified from Shell) the default archive file name stored in global variable *TAPE* is used.

The *bfactor* dictates the blocking factor the tape was written with. If 0, or unspecified from the shell, a default of 20 is used.

The *verbose* argument is a boolean, if set to 1, will cause informative messages to be printed to standard error whenever an action is taken, otherwise, only errors are reported.

All informative and error message are printed to standard error.

There is no way to selectively extract tar archives with this utility. It extracts entire archives.

**SEE ALSO****tarLib**

---

**tarToc()****NAME****tarToc()** – display all contents of a tar formatted tape**SYNOPSIS**

```
STATUS tarToc
  (
    char * tape,           /* tape device name */
    int bfactor          /* requested blocking factor */
  )
```

**DESCRIPTION**

This is a UNIX-tar compatible utility that displays entire file hierarchies from tar-formatted media, *e.g.* tape.

The *tape* argument may be any tape device name or file name that contains a tar formatted archive. If *tape* is equal "-", standard input is used. If *tape* is NULL (or unspecified from Shell) the default archive file name stored in global variable *TAPE* is used.

The *bfactor* dictates the blocking factor the tape was written with. If 0, or unspecified from Shell, default of 20 is used.

Archive contents are displayed on standard output, while all informative and error message are printed to standard error.

**SEE ALSO****tarLib**

---

## taskActivate()

**NAME** `taskActivate()` – activate a task that has been initialized

**SYNOPSIS**

```
STATUS taskActivate
(
    int tid                /* task ID of task to activate */
)
```

**DESCRIPTION** This routine activates tasks created by `taskInit()`. Without activation, a task is ineligible for CPU allocation by the scheduler.

The *tid* (task ID) argument is simply the address of the `WIND_TCB` for the task (the `taskInit()` *pTcb* argument), cast to an integer:

```
tid = (int) pTcb;
```

The `taskSpawn()` routine is built from `taskActivate()` and `taskInit()`. Tasks created by `taskSpawn()` do not require explicit task activation.

**RETURNS** `OK`, or `ERROR` if the task cannot be activated.

**SEE ALSO** `taskLib`, `taskInit()`

---

## taskCreateHookAdd()

**NAME** `taskCreateHookAdd()` – add a routine to be called at every task create

**SYNOPSIS**

```
STATUS taskCreateHookAdd
(
    FUNCPTR createHook    /* routine to be called when a task is created */
)
```

**DESCRIPTION** This routine adds a specified routine to a list of routines that will be called whenever a task is created. The routine should be declared as follows:

```
void createHook
(
    WIND_TCB *pNewTcb     /* pointer to new task's TCB */
)
```

**RETURNS** OK, or ERROR if the table of task create routines is full.

**SEE ALSO** taskHookLib, taskCreateHookDelete()

---

## taskCreateHookDelete()

**NAME** taskCreateHookDelete() – delete a previously added task create routine

**SYNOPSIS**

```
STATUS taskCreateHookDelete
(
    FUNCPTR createHook      /* routine to be deleted from list */
)
```

**DESCRIPTION** This routine removes a specified routine from the list of routines to be called at each task create.

**RETURNS** OK, or ERROR if the routine is not in the table of task create routines.

**SEE ALSO** taskHookLib, taskCreateHookAdd()

---

## taskCreateHookShow()

**NAME** taskCreateHookShow() – show the list of task create routines

**SYNOPSIS** void taskCreateHookShow (void)

**DESCRIPTION** This routine shows all the task create routines installed in the task create hook table, in the order in which they were installed.

**RETURNS** N/A

**SEE ALSO** taskHookShow, taskCreateHookAdd()

---

## taskDelay()

|                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
|--------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>        | <b>taskDelay()</b> – delay a task from executing                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| <b>SYNOPSIS</b>    | <pre><b>STATUS</b> taskDelay (     int ticks                /* number of ticks to delay task */ )</pre>                                                                                                                                                                                                                                                                                                                                                                            |
| <b>DESCRIPTION</b> | <p>This routine causes the calling task to relinquish the CPU for the duration specified (in ticks). This is commonly referred to as manual rescheduling, but it is also useful when waiting for some external condition that does not have an interrupt associated with it.</p> <p>If the calling task receives a signal that is not being blocked or ignored, <b>taskDelay()</b> returns <b>ERROR</b> and sets <b>errno</b> to <b>EINTR</b> after the signal handler is run.</p> |
| <b>RETURNS</b>     | OK, or <b>ERROR</b> if called from interrupt level or if the calling task receives a signal that is not blocked or ignored.                                                                                                                                                                                                                                                                                                                                                        |
| <b>ERRNO</b>       | <b>S_intLib_NOT_ISR_CALLABLE</b> , <b>EINTR</b>                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| <b>SEE ALSO</b>    | <b>taskLib</b>                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |

---

## taskDelete()

|                    |                                                                                                                                                                                                                                                                                                                    |
|--------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>        | <b>taskDelete()</b> – delete a task                                                                                                                                                                                                                                                                                |
| <b>SYNOPSIS</b>    | <pre><b>STATUS</b> taskDelete (     int tid                /* task ID of task to delete */ )</pre>                                                                                                                                                                                                                 |
| <b>DESCRIPTION</b> | <p>This routine causes a specified task to cease to exist and deallocates the stack and <b>WIND_TCB</b> memory resources. Upon deletion, all routines specified by <b>taskDeleteHookAdd()</b> will be called in the context of the deleting task. This routine is the companion routine to <b>taskSpawn()</b>.</p> |
| <b>RETURNS</b>     | OK, or <b>ERROR</b> if the task cannot be deleted.                                                                                                                                                                                                                                                                 |

**ERRNO** S\_intLib\_NOT\_ISR\_CALLABLE, S\_objLib\_OBJ\_DELETED, S\_objLib\_OBJ\_UNAVAILABLE, S\_objLib\_OBJ\_ID\_ERROR

**SEE ALSO** taskLib, excLib, taskDeleteHookAdd(), taskSpawn(), *VxWorks Programmer's Guide: Basic OS*

---

## taskDeleteForce()

**NAME** taskDeleteForce() – delete a task without restriction

**SYNOPSIS**

```
STATUS taskDeleteForce  
(  
    int tid                /* task ID of task to delete */  
)
```

**DESCRIPTION** This routine deletes a task even if the task is protected from deletion. It is similar to **taskDelete()**. Upon deletion, all routines specified by **taskDeleteHookAdd()** will be called in the context of the deleting task.

---

**WARNING:** This routine is intended as a debugging aid, and is generally inappropriate for applications. Disregarding a task's deletion protection could leave the system in an unstable state or lead to system deadlock.

---

The system does not protect against simultaneous **taskDeleteForce()** calls. Such a situation could leave the system in an unstable state.

**RETURNS** OK, or ERROR if the task cannot be deleted.

**ERRNO** S\_intLib\_NOT\_ISR\_CALLABLE, S\_objLib\_OBJ\_DELETED, S\_objLib\_OBJ\_UNAVAILABLE, S\_objLib\_OBJ\_ID\_ERROR

**SEE ALSO** taskLib, taskDeleteHookAdd(), taskDelete()

---

## taskDeleteHookAdd()

- NAME** `taskDeleteHookAdd()` – add a routine to be called at every task delete
- SYNOPSIS**
- ```
STATUS taskDeleteHookAdd
(
    FUNCPTR deleteHook    /* routine to be called when a task is deleted */
)
```
- DESCRIPTION** This routine adds a specified routine to a list of routines that will be called whenever a task is deleted. The routine should be declared as follows:
- ```
void deleteHook
(
    WIND_TCB *pTcb        /* pointer to deleted task's WIND_TCB */
)
```
- RETURNS** OK, or **ERROR** if the table of task delete routines is full.
- SEE ALSO** `taskHookLib`, `taskDeleteHookDelete()`

---

## taskDeleteHookDelete()

- NAME** `taskDeleteHookDelete()` – delete a previously added task delete routine
- SYNOPSIS**
- ```
STATUS taskDeleteHookDelete
(
    FUNCPTR deleteHook    /* routine to be deleted from list */
)
```
- DESCRIPTION** This routine removes a specified routine from the list of routines to be called at each task delete.
- RETURNS** OK, or **ERROR** if the routine is not in the table of task delete routines.
- SEE ALSO** `taskHookLib`, `taskDeleteHookAdd()`

## **taskDeleteHookShow()**

<b>NAME</b>	<b>taskDeleteHookShow()</b> – show the list of task delete routines
<b>SYNOPSIS</b>	<b>void taskDeleteHookShow (void)</b>
<b>DESCRIPTION</b>	This routine shows all the delete routines installed in the task delete hook table, in the order in which they were installed. Note that the delete routines will be run in reverse of the order in which they were installed.
<b>RETURNS</b>	N/A
<b>SEE ALSO</b>	<b>taskHookShow, taskDeleteHookAdd()</b>

---

## **taskHookInit()**

<b>NAME</b>	<b>taskHookInit()</b> – initialize task hook facilities
<b>SYNOPSIS</b>	<b>void taskHookInit (void)</b>
<b>DESCRIPTION</b>	This routine is a NULL routine called to configure the task hook package into the system. It is called automatically if the configuration macro <b>INCLUDE_TASK_HOOKS</b> is defined.
<b>RETURNS</b>	N/A
<b>SEE ALSO</b>	<b>taskHookLib</b>

---

## taskHookShowInit()

- NAME** `taskHookShowInit()` – initialize the task hook show facility
- SYNOPSIS** `void taskHookShowInit (void)`
- DESCRIPTION** This routine links the task hook show facility into the VxWorks system. It is called automatically when the task hook show facility is configured into VxWorks using either of the following methods:
- If you use the configuration header files, define `INCLUDE_SHOW_ROUTINES` in `config.h`.
  - If you use the Tornado project facility, select `INCLUDE_TASK_HOOK_SHOW`.
- RETURNS** N/A
- SEE ALSO** `taskHookShow`

---

## taskIdDefault()

- NAME** `taskIdDefault()` – set the default task ID
- SYNOPSIS**
- ```
int taskIdDefault
(
    int tid                /* user supplied task ID; if 0, return default */
)
```
- DESCRIPTION** This routine maintains a global default task ID. This ID is used by libraries that want to allow a task ID argument to take on a default value if the user did not explicitly supply one.
- If `tid` is not zero (*i.e.*, the user did specify a task ID), the default ID is set to that value, and that value is returned. If `tid` is zero (*i.e.*, the user did not specify a task ID), the default ID is not changed and its value is returned. Thus the value returned is always the last task ID the user specified.
- RETURNS** The most recent non-zero task ID.
- SEE ALSO** `taskInfo`, `dbgLib`, *VxWorks Programmer's Guide: Target Shell*, `windsh`, *Tornado User's Guide: Shell*

---

## taskIdListGet()

|                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
|--------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>        | <b>taskIdListGet()</b> – get a list of active task IDs                                                                                                                                                                                                                                                                                                                                                                                              |
| <b>SYNOPSIS</b>    | <pre>int taskIdListGet (     int idList[],          /* array of task IDs to be filled in */     int maxTasks          /* max tasks idList can accommodate */ )</pre>                                                                                                                                                                                                                                                                                |
| <b>DESCRIPTION</b> | <p>This routine provides the calling task with a list of all active tasks. An unsorted list of task IDs for no more than <i>maxTasks</i> tasks is put into <i>idList</i>.</p> <hr/> <p><b>WARNING:</b> Kernel rescheduling is disabled with <b>taskLock()</b> while tasks are filled into the <i>idList</i>. There is no guarantee that all the tasks are valid or that new tasks have not been created by the time this routine returns.</p> <hr/> |
| <b>RETURNS</b>     | The number of tasks put into the ID list.                                                                                                                                                                                                                                                                                                                                                                                                           |
| <b>SEE ALSO</b>    | <b>taskInfo</b>                                                                                                                                                                                                                                                                                                                                                                                                                                     |

---

## taskIdSelf()

|                    |                                                                                                                     |
|--------------------|---------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>        | <b>taskIdSelf()</b> – get the task ID of a running task                                                             |
| <b>SYNOPSIS</b>    | <pre>int taskIdSelf (void)</pre>                                                                                    |
| <b>DESCRIPTION</b> | <p>This routine gets the task ID of the calling task. The task ID will be invalid if called at interrupt level.</p> |
| <b>RETURNS</b>     | The task ID of the calling task.                                                                                    |
| <b>SEE ALSO</b>    | <b>taskLib</b>                                                                                                      |

---

## taskIdVerify()

**NAME** `taskIdVerify()` – verify the existence of a task

**SYNOPSIS**

```
STATUS taskIdVerify
(
    int tid                /* task ID */
)
```

**DESCRIPTION** This routine verifies the existence of a specified task by validating the specified ID as a task ID. Note that an exception occurs if the task ID parameter points to an address not located in physical memory.

**RETURNS** OK, or ERROR if the task ID is invalid.

**ERRNO** S\_objLib\_OBJ\_ID\_ERROR

**SEE ALSO** taskLib

---

## taskInfoGet()

**NAME** `taskInfoGet()` – get information about a task

**SYNOPSIS**

```
STATUS taskInfoGet
(
    int      tid,          /* ID of task for which to get info */
    TASK_DESC * pTaskDesc /* task descriptor to be filled in */
)
```

**DESCRIPTION** This routine fills in a specified task descriptor (TASK\_DESC) for a specified task. The information in the task descriptor is, for the most part, a copy of information kept in the task control block (WIND\_TCB). The TASK\_DESC structure is useful for common information and avoids dealing directly with the unwieldy WIND\_TCB.

---

**NOTE:** Examination of WIND\_TCBs should be restricted to debugging aids.

---

**RETURNS** OK, or ERROR if the task ID is invalid.

**SEE ALSO** taskShow

**taskInit()****taskInit()**

**NAME** `taskInit()` – initialize a task with a stack at a specified address

**SYNOPSIS**

```

STATUS taskInit
(
    WIND_TCB * pTcb,           /* address of new task's TCB */
    char *     name,          /* name of new task (stored at pStackBase) */
    int        priority,      /* priority of new task */
    int        options,       /* task option word */
    char *     pStackBase,    /* base of new task's stack */
    int        stackSize,     /* size (bytes) of stack needed */
    FUNCPTR   entryPt,       /* entry point of new task */
    int        arg1,          /* first of ten task args to pass to func */
    int        arg2,
    int        arg3,
    int        arg4,
    int        arg5,
    int        arg6,
    int        arg7,
    int        arg8,
    int        arg9,
    int        arg10
)

```

**DESCRIPTION** This routine initializes user-specified regions of memory for a task stack and control block instead of allocating them from memory as `taskSpawn()` does. This routine will utilize the specified pointers to the `WIND_TCB` and stack as the components of the task. This allows, for example, the initialization of a static `WIND_TCB` variable. It also allows for special stack positioning as a debugging aid.

As in `taskSpawn()`, a task may be given a name. While `taskSpawn()` automatically names unnamed tasks, `taskInit()` permits the existence of tasks without names. The task ID required by other task routines is simply the address `pTcb`, cast to an integer.

Note that the task stack may grow up or down from `pStackBase`, depending on the target architecture.

Other arguments are the same as in `taskSpawn()`. Unlike `taskSpawn()`, `taskInit()` does not activate the task. This must be done by calling `taskActivate()` after calling `taskInit()`.

Normally, tasks should be started using `taskSpawn()` rather than `taskInit()`, except when additional control is required for task memory allocation or a separate task activation is desired.

**RETURNS** OK, or ERROR if the task cannot be initialized.

**ERRNO** S\_intLib\_NOT\_ISR\_CALLABLE, S\_objLib\_OBJ\_ID\_ERROR, S\_taskLib\_ILLEGAL\_PRIORITY

**SEE ALSO** taskLib, taskActivate(), taskSpawn()

---

## taskIsReady()

**NAME** taskIsReady() – check if a task is ready to run

**SYNOPSIS**

```
BOOL taskIsReady
(
    int tid                /* task ID */
)
```

**DESCRIPTION** This routine tests the status field of a task to determine if it is ready to run.

**RETURNS** TRUE if the task is ready, otherwise FALSE.

**SEE ALSO** taskInfo

---

## taskIsSuspended()

**NAME** taskIsSuspended() – check if a task is suspended

**SYNOPSIS**

```
BOOL taskIsSuspended
(
    int tid                /* task ID */
)
```

**DESCRIPTION** This routine tests the status field of a task to determine if it is suspended.

**RETURNS** TRUE if the task is suspended, otherwise FALSE.

**SEE ALSO** taskInfo

## taskLock()

|                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
|--------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>        | <b>taskLock()</b> – disable task rescheduling                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| <b>SYNOPSIS</b>    | <b>STATUS</b> <b>taskLock</b> (void)                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| <b>DESCRIPTION</b> | <p>This routine disables task context switching. The task that calls this routine will be the only task that is allowed to execute, unless the task explicitly gives up the CPU by making itself no longer ready. Typically this call is paired with <b>taskUnlock()</b>; together they surround a critical section of code. These preemption locks are implemented with a counting variable that allows nested preemption locks. Preemption will not be unlocked until <b>taskUnlock()</b> has been called as many times as <b>taskLock()</b>.</p> <p>This routine does not lock out interrupts; use <b>intLock()</b> to lock out interrupts.</p> <p>A <b>taskLock()</b> is preferable to <b>intLock()</b> as a means of mutual exclusion, because interrupt lock-outs add interrupt latency to the system.</p> <p>A <b>semTake()</b> is preferable to <b>taskLock()</b> as a means of mutual exclusion, because preemption lock-outs add preemptive latency to the system.</p> <p>The <b>taskLock()</b> routine is not callable from interrupt service routines.</p> |
| <b>RETURNS</b>     | OK or ERROR.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| <b>ERRNO</b>       | S_objLib_OBJ_ID_ERROR, S_intLib_NOT_ISR_CALLABLE                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
| <b>SEE ALSO</b>    | <b>taskLib</b> , <b>taskUnlock()</b> , <b>intLock()</b> , <b>taskSafe()</b> , <b>semTake()</b>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |

---

## taskName()

|                    |                                                                                                                                                             |
|--------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>        | <b>taskName()</b> – get the name associated with a task ID                                                                                                  |
| <b>SYNOPSIS</b>    | <pre>char *taskName (     int tid                /* ID of task whose name is to be found */ )</pre>                                                         |
| <b>DESCRIPTION</b> | <p>This routine returns a pointer to the name of a task of a specified ID, if the task has a name. If the task has no name, it returns an empty string.</p> |

**RETURNS** A pointer to the task name, or NULL if the task ID is invalid.

**SEE ALSO** `taskInfo`

## taskNameToId()

**NAME** `taskNameToId()` – look up the task ID associated with a task name

**SYNOPSIS**

```
int taskNameToId
(
    char * name           /* task name to look up */
)
```

**DESCRIPTION** This routine returns the ID of the task matching a specified name. Referencing a task in this way is inefficient, since it involves a search of the task list.

**RETURNS** The task ID, or **ERROR** if the task is not found.

**ERRNO** `S_taskLib_NAME_NOT_FOUND`

**SEE ALSO** `taskInfo`

## taskOptionsGet()

**NAME** `taskOptionsGet()` – examine task options

**SYNOPSIS**

```
STATUS taskOptionsGet
(
    int tid,              /* task ID */
    int * pOptions       /* task's options */
)
```

**DESCRIPTION** This routine gets the current execution options of the specified task. The option bits returned by this routine indicate the following modes:

**VX\_FP\_TASK**  
 execute with floating-point coprocessor support.

T

**VX\_PRIVATE\_ENV**  
include private environment support (see **envLib**).

**VX\_NO\_STACK\_FILL**  
do not fill the stack for use by **checkstack()**.

**VX\_UNBREAKABLE**  
do not allow breakpoint debugging.

For definitions, see **taskLib.h**.

**RETURNS** OK, or **ERROR** if the task ID is invalid.

**SEE ALSO** **taskInfo**, **taskOptionsSet()**

---

## **taskOptionsSet()**

**NAME** **taskOptionsSet()** – change task options

**SYNOPSIS**

```
STATUS taskOptionsSet  
(  
    int tid,                /* task ID */  
    int mask,               /* bit mask of option bits to unset */  
    int newOptions          /* bit mask of option bits to set */  
)
```

**DESCRIPTION** This routine changes the execution options of a task. The only option that can be changed after a task has been created is:

**VX\_UNBREAKABLE**  
do not allow breakpoint debugging.

For definitions, see **taskLib.h**.

**RETURNS** OK, or **ERROR** if the task ID is invalid.

**SEE ALSO** **taskInfo**, **taskOptionsGet()**

---

## taskPriorityGet()

**NAME** taskPriorityGet() – examine the priority of a task

**SYNOPSIS**

```
STATUS taskPriorityGet
(
    int    tid,                /* task ID */
    int *  pPriority           /* return priority here */
)
```

**DESCRIPTION** This routine determines the current priority of a specified task. The current priority is copied to the integer pointed to by *pPriority*.

**RETURNS** OK, or ERROR if the task ID is invalid.

**ERRNO** S\_objLib\_OBJ\_ID\_ERROR

**SEE ALSO** taskLib, taskPrioritySet()

---

## taskPrioritySet()

**NAME** taskPrioritySet() – change the priority of a task

**SYNOPSIS**

```
STATUS taskPrioritySet
(
    int tid,                /* task ID */
    int newPriority         /* new priority */
)
```

**DESCRIPTION** This routine changes a task's priority to a specified priority. Priorities range from 0, the highest priority, to 255, the lowest priority.

**RETURNS** OK, or ERROR if the task ID is invalid.

**ERRNO** S\_taskLib\_ILLEGAL\_PRIORITY, S\_objLib\_OBJ\_ID\_ERROR

**SEE ALSO** taskLib, taskPriorityGet()

---

## taskRegsGet()

**NAME** taskRegsGet() – get a task’s registers from the TCB

**SYNOPSIS**

```
STATUS taskRegsGet
(
    int      tid,          /* task ID */
    REG_SET * pRegs      /* put register contents here */
)
```

**DESCRIPTION** This routine gathers task information kept in the TCB. It copies the contents of the task’s registers to the register structure *pRegs*.

---

**NOTE:** This routine only works well if the task is known to be in a stable, non-executing state. Self-examination, for instance, is not advisable, as results are unpredictable.

---

**RETURNS** OK, or ERROR if the task ID is invalid.

**SEE ALSO** taskInfo, taskSuspend(), taskRegsSet()

---

## taskRegsSet()

**NAME** taskRegsSet() – set a task’s registers

**SYNOPSIS**

```
STATUS taskRegsSet
(
    int      tid,          /* task ID */
    REG_SET * pRegs      /* get register contents from here */
)
```

**DESCRIPTION** This routine loads a specified register set *pRegs* into a specified task’s TCB.

---

**NOTE:** This routine only works well if the task is known not to be in the ready state. Suspending the task before changing the register set is recommended.

---

**RETURNS** OK, or ERROR if the task ID is invalid.

**SEE ALSO** taskInfo, taskSuspend(), taskRegsGet()

---

## taskRegsShow( )

**NAME** `taskRegsShow( )` – display the contents of a task’s registers

**SYNOPSIS**

```
void taskRegsShow
(
    int tid                /* task ID */
)
```

**DESCRIPTION** This routine displays the register contents of a specified task on standard output.

**EXAMPLE** The following example displays the register of the shell task (68000 family):

```
-> taskRegsShow (taskNameToId ("tShell"))
d0 = 0 d1 = 0 d2 = 578fe d3 = 1
d4 = 3e84e1 d5 = 3e8568 d6 = 0 d7 = ffffffff
a0 = 0 a1 = 0 a2 = 4f06c a3 = 578d0
a4 = 3fffc4 a5 = 0 fp = 3e844c sp = 3e842c
sr = 3000 pc = 4f0f2
value = 0 = 0x0
```

**RETURNS** N/A

**SEE ALSO** `taskShow`

---

## taskRestart( )

**NAME** `taskRestart( )` – restart a task

**SYNOPSIS**

```
STATUS taskRestart
(
    int tid                /* task ID of task to restart */
)
```

**DESCRIPTION** This routine “restarts” a task. The task is first terminated, and then re-initialized with the same ID, priority, options, original entry point, stack size, and parameters it had when it was terminated. Self-restarting of a calling task is performed by the exception task. The shell utilizes this routine to restart itself when aborted.

## **taskResume()**

---

**NOTE:** If the task has modified any of its start-up parameters, the restarted task will start with the changed values.

---

**RETURNS** OK, or ERROR if the task ID is invalid or the task could not be restarted.

**ERRNO** S\_intLib\_NOT\_ISR\_CALLABLE, S\_objLib\_OBJ\_DELETED, S\_objLib\_OBJ\_UNAVAILABLE, S\_objLib\_OBJ\_ID\_ERROR, S\_smObjLib\_NOT\_INITIALIZED, S\_memLib\_NOT\_ENOUGH\_MEMORY, S\_memLib\_BLOCK\_ERROR, S\_taskLib\_ILLEGAL\_PRIORITY

**SEE ALSO** taskLib

---

## **taskResume()**

**NAME** taskResume() – resume a task

**SYNOPSIS**

```
STATUS taskResume  
(  
    int tid                /* task ID of task to resume */  
)
```

**DESCRIPTION** This routine resumes a specified task. Suspension is cleared, and the task operates in the remaining state.

**RETURNS** OK, or ERROR if the task cannot be resumed.

**ERRNO** S\_objLib\_OBJ\_ID\_ERROR

**SEE ALSO** taskLib

---

## **taskSafe()**

**NAME** taskSafe() – make the calling task safe from deletion

**SYNOPSIS**

```
STATUS taskSafe (void)
```

- DESCRIPTION** This routine protects the calling task from deletion. Tasks that attempt to delete a protected task will block until the task is made unsafe, using **taskUnsafe()**. When a task becomes unsafe, the deleter will be unblocked and allowed to delete the task.
- The **taskSafe()** primitive utilizes a count to keep track of nested calls for task protection. When nesting occurs, the task becomes unsafe only after the outermost **taskUnsafe()** is executed.
- RETURNS** OK.
- SEE ALSO** **taskLib**, **taskUnsafe()**, *VxWorks Programmer's Guide: Basic OS*

---

## taskShow()

- NAME** **taskShow()** – display task information from TCBs
- SYNOPSIS**
- ```
STATUS taskShow
(
    int tid,                /* task ID */
    int level               /* 0 = summary, 1 = details, 2 = all tasks */
)
```
- DESCRIPTION** This routine displays the contents of a task control block (TCB) for a specified task. If *level* is 1, it also displays task options and registers. If *level* is 2, it displays all tasks.
- The TCB display contains the following fields:

Field	Meaning
NAME	Task name
ENTRY	Symbol name or address where task began execution
TID	Task ID
PRI	Priority
STATUS	Task status, as formatted by <b>taskStatusString()</b>
PC	Program counter
SP	Stack pointer
ERRNO	Most recent error code for this task
DELAY	If task is delayed, number of clock ticks remaining in delay (0 otherwise)

- EXAMPLE** The following example shows the TCB contents for the shell task:

```
-> taskShow tShell, 1
```

**taskShowInit()**

```

      NAME          ENTRY      TID     PRI  STATUS      PC          SP      ERRNO  DELAY
-----
tShell    _shell      20efcac  1  READY      201dc90    20ef980    0      0
stack: base 0x20efcac  end 0x20ed59c  size 9532  high 1452  margin 8080
options: 0x1e
VX_UNBREAKABLE      VX_DEALLOC_STACK      VX_FP_TASK      VX_STDIO
VxWorks Events
-----
Events Pended on      : Not Pended
Received Events       : 0x0
Options               : N/A

D0 =      0    D4 =      0    A0 =      0    A4 =      0
D1 =      0    D5 =      0    A1 =      0    A5 = 203a084  SR =      3000
D2 =      0    D6 =      0    A2 =      0    A6 = 20ef9a0  PC = 2038614
D3 =      0    D7 =      0    A3 =      0    A7 = 20ef980
value = 34536868 = 0x20efda4

```

**RETURNS** N/A**SEE ALSO** **taskShow**, **taskStatusString()**, *VxWorks Programmer's Guide: Target Shell*, **windsh**, *Tornado User's Guide: Shell*

---

**taskShowInit()****NAME** **taskShowInit()** – initialize the task show routine facility**SYNOPSIS** **void taskShowInit (void)****DESCRIPTION** This routine links the task show routines into the VxWorks system. It is called automatically when the task show facility is configured into VxWorks using either of the following methods:

- If you use the configuration header files, define **INCLUDE\_SHOW\_ROUTINES** in **config.h**.
- If you use the Tornado project facility, select **INCLUDE\_TASK\_SHOW**.

**RETURNS** N/A**SEE ALSO** **taskShow**

---

## taskSpawn()

**NAME** taskSpawn() – spawn a task

**SYNOPSIS**

```
int taskSpawn
(
    char * name,          /* name of new task (stored at pStackBase) */
    int priority,        /* priority of new task */
    int options,         /* task option word */
    int stackSize,       /* size (bytes) of stack needed plus name */
    FUNCPTR entryPt,     /* entry point of new task */
    int arg1,            /* 1st of 10 req'd task args to pass to func */
    int arg2,
    int arg3,
    int arg4,
    int arg5,
    int arg6,
    int arg7,
    int arg8,
    int arg9,
    int arg10
)
```

**DESCRIPTION** This routine creates and activates a new task with a specified priority and options and returns a system-assigned ID. See **taskInit()** and **taskActivate()** for the building blocks of this routine.

A task may be assigned a name as a debugging aid. This name will appear in displays generated by various system information facilities such as **i()**. The name may be of arbitrary length and content, but the current VxWorks convention is to limit task names to ten characters and prefix them with a "t". If *name* is specified as NULL, an ASCII name will be assigned to the task of the form "tn" where *n* is an integer which increments as new tasks are spawned.

The only resource allocated to a spawned task is a stack of a specified size *stackSize*, which is allocated from the system memory partition. Stack size should be an even integer. A task control block (TCB) is carved from the stack, as well as any memory required by the task name. The remaining memory is the task's stack and every byte is filled with the value 0xEE for the **checkStack()** facility. See the manual entry for **checkStack()** for stack-size checking aids.

The entry address *entryPt* is the address of the "main" routine of the task. The routine will be called once the C environment has been set up. The specified routine will be called with the ten given arguments. Should the specified main routine return, a call to **exit()** will automatically be made.

**taskSRInit()**

Note that ten (and only ten) arguments must be passed for the spawned function.

Bits in the options argument may be set to run with the following modes:

**VX\_FP\_TASK** (0x0008)

execute with floating-point coprocessor support. A task which performs floating point operations or calls any functions which either return or take a floating point value as arguments must be created with this option. Some routines perform floating point operations internally. The VxWorks documentation for these clearly state the need to use the **VX\_FP\_TASK** option.

**VX\_PRIVATE\_ENV** (0x0080)

include private environment support (see **envLib**).

**VX\_NO\_STACK\_FILL** (0x0100)

do not fill the stack for use by **checkStack()**.

**VX\_UNBREAKABLE** (0x0002)

do not allow breakpoint debugging.

See the definitions in **taskLib.h**.

**RETURNS**

The task ID, or **ERROR** if memory is insufficient or the task cannot be created.

**ERRNO**

**S\_intLib\_NOT\_ISR\_CALLABLE**, **S\_objLib\_OBJ\_ID\_ERROR**, **S\_smObjLib\_NOT\_INITIALIZED**, **S\_memLib\_NOT\_ENOUGH\_MEMORY**, **S\_memLib\_BLOCK\_ERROR**, **S\_taskLib\_ILLEGAL\_PRIORITY**

**SEE ALSO**

**taskLib**, **taskInit()**, **taskActivate()**, **sp()**, *VxWorks Programmer's Guide: Basic OS*

**taskSRInit()****NAME**

**taskSRInit()** – initialize the default task status register (MIPS)

**SYNOPSIS**

```
ULONG taskSRInit
(
    ULONG newSRValue           /* new default task status register */
)
```

**DESCRIPTION**

This routine sets the default status register for system-wide tasks. All tasks are spawned with the status register set to this value; thus, it must be called before **kernelInit()**.

**RETURNS**

The previous value of the default status register.

**SEE ALSO**

**taskArchLib**

---

## taskSRSet()

**NAME** `taskSRSet()` – set the task status register (68K, MIPS, x86)

**SYNOPSIS**

```
STATUS taskSRSet
(
    int    tid,           /* task ID */
    UINT16 sr            /* new SR */
)
```

**DESCRIPTION** This routine sets the status register of a task that is not running (*i.e.*, the TCB must not be that of the calling task). Debugging facilities use this routine to set the trace bit in the status register of a task that is being single-stepped.

x86:

The second parameter represents EFLAGS register and the size is 32 bit.

**RETURNS** OK, or ERROR if the task ID is invalid.

**SEE ALSO** `taskArchLib`

---

## taskStatusString()

**NAME** `taskStatusString()` – get a task's status as a string

**SYNOPSIS**

```
STATUS taskStatusString
(
    int    tid,           /* task to get string for */
    char * pString       /* where to return string */
)
```

**DESCRIPTION** This routine decipheres the WIND task status word in the TCB for a specified task, and copies the appropriate string to *pString*.

The formatted string is one of the following:

String	Meaning
READY	Task is not waiting for any resource other than the CPU.
PEND	Task is blocked due to the unavailability of some resource.
DELAY	Task is asleep for some duration.
SUSPEND	Task is unavailable for execution (but not suspended, delayed, or pending).

**taskSuspend()**

String	Meaning
DELAY+S	Task is both delayed and suspended.
PEND+S	Task is both pended and suspended.
PEND+T	Task is pended with a timeout.
PEND+S+T	Task is pended with a timeout, and also suspended.
...+I	Task has inherited priority (+I may be appended to any string above).
DEAD	Task no longer exists.

**EXAMPLE**

```

-> taskStatusString (taskNameToId ("tShell"), xx=malloc (10))
new symbol "xx" added to symbol table.
value = 0 = 0x0
-> printf ("shell status = <%s>\n", xx)
shell status = <READY>
value = 2 = 0x2

```

**RETURNS** OK, or ERROR if the task ID is invalid.

**SEE ALSO** taskShow

---

## taskSuspend()

**NAME** taskSuspend() – suspend a task

**SYNOPSIS**

```

STATUS taskSuspend
(
    int tid                /* task ID of task to suspend */
)

```

**DESCRIPTION** This routine suspends a specified task. A task ID of zero results in the suspension of the calling task. Suspension is additive, thus tasks can be delayed and suspended, or pended and suspended. Suspended, delayed tasks whose delays expire remain suspended. Likewise, suspended, pended tasks that unblock remain suspended only.

Care should be taken with asynchronous use of this facility. The specified task is suspended regardless of its current state. The task could, for instance, have mutual exclusion to some system resource, such as the network \* or system memory partition. If suspended during such a time, the facilities engaged are unavailable, and the situation often ends in deadlock.

This routine is the basis of the debugging and exception handling packages. However, as a synchronization mechanism, this facility should be rejected in favor of the more general semaphore facility.

**RETURNS** OK, or **ERROR** if the task cannot be suspended.

**ERRNO** S\_objLib\_OBJ\_ID\_ERROR

**SEE ALSO** taskLib

---

## taskSwitchHookAdd()

**NAME** taskSwitchHookAdd() – add a routine to be called at every task switch

**SYNOPSIS**

```
STATUS taskSwitchHookAdd
(
    FUNCPTR switchHook      /* routine to be called at every task switch */
)
```

**DESCRIPTION** This routine adds a specified routine to a list of routines that will be called at every task switch. The routine should be declared as follows:

```
void switchHook
(
    WIND_TCB *pOldTcb,      /* pointer to old task's WIND_TCB */
    WIND_TCB *pNewTcb      /* pointer to new task's WIND_TCB */
)
```

**NOTE** User-installed switch hooks are called within the kernel context. Therefore, switch hooks do not have access to all VxWorks facilities. The following routines can be called from within a task switch hook:

Library	Routines
bLib	All routines
fppArchLib	fppSave(), fppRestore()
intLib	intContext(), intCount(), intVecSet(), intVecGet()
lstLib	All routines
mathALib	All routines, if fppSave()/fppRestore() are used
rngLib	All routines except rngCreate()
taskLib	taskIdVerify(), taskIdDefault(), taskIsReady(), taskIsSuspended(), taskTcb()
vxLib	vxTas()

**RETURNS** OK, or **ERROR** if the table of task switch routines is full.

**SEE ALSO** taskHookLib, taskSwitchHookDelete()

## **taskSwitchHookDelete()**

<b>NAME</b>	<b>taskSwitchHookDelete()</b> – delete a previously added task switch routine
<b>SYNOPSIS</b>	<pre><b>STATUS</b> taskSwitchHookDelete (     <b>FUNCPTR</b> switchHook      /* routine to be deleted from list */ )</pre>
<b>DESCRIPTION</b>	This routine removes the specified routine from the list of routines to be called at each task switch.
<b>RETURNS</b>	OK, or ERROR if the routine is not in the table of task switch routines.
<b>SEE ALSO</b>	taskHookLib, taskSwitchHookAdd()

---

## **taskSwitchHookShow()**

<b>NAME</b>	<b>taskSwitchHookShow()</b> – show the list of task switch routines
<b>SYNOPSIS</b>	<pre><b>void</b> taskSwitchHookShow (<b>void</b>)</pre>
<b>DESCRIPTION</b>	This routine shows all the switch routines installed in the task switch hook table, in the order in which they were installed.
<b>RETURNS</b>	N/A
<b>SEE ALSO</b>	taskHookShow, taskSwitchHookAdd()

---

## taskTcb()

**NAME** `taskTcb()` – get the task control block for a task ID

**SYNOPSIS**

```
WIND_TCB *taskTcb
(
    int tid                /* task ID */
)
```

**DESCRIPTION** This routine returns a pointer to the task control block (`WIND_TCB`) for a specified task. Although all task state information is contained in the TCB, users must not modify it directly. To change registers, for instance, use `taskRegsSet()` and `taskRegsGet()`.

**RETURNS** A pointer to a `WIND_TCB`, or `NULL` if the task ID is invalid.

**ERRNO** `S_objLib_OBJ_ID_ERROR`

**SEE ALSO** `taskLib`

---

## taskUnlock()

**NAME** `taskUnlock()` – enable task rescheduling

**SYNOPSIS** `STATUS taskUnlock (void)`

**DESCRIPTION** This routine decrements the preemption lock count. Typically this call is paired with `taskLock()` and concludes a critical section of code. Preemption will not be unlocked until `taskUnlock()` has been called as many times as `taskLock()`. When the lock count is decremented to zero, any tasks that were eligible to preempt the current task will execute.

The `taskUnlock()` routine is not callable from interrupt service routines.

**RETURNS** `OK` or `ERROR`.

**ERRNO** `S_intLib_NOT_ISR_CALLABLE`

**SEE ALSO** `taskLib`, `taskLock()`

---

## taskUnsafe()

<b>NAME</b>	<b>taskUnsafe()</b> – make the calling task unsafe from deletion
<b>SYNOPSIS</b>	<b>STATUS</b> <b>taskUnsafe</b> (void)
<b>DESCRIPTION</b>	<p>This routine removes the calling task's protection from deletion. Tasks that attempt to delete a protected task will block until the task is unsafe. When a task becomes unsafe, the deleter will be unblocked and allowed to delete the task.</p> <p>The <b>taskUnsafe()</b> primitive utilizes a count to keep track of nested calls for task protection. When nesting occurs, the task becomes unsafe only after the outermost <b>taskUnsafe()</b> is executed.</p>
<b>RETURNS</b>	OK.
<b>SEE ALSO</b>	<b>taskLib</b> , <b>taskSafe()</b> , <i>VxWorks Programmer's Guide: Basic OS</i>

---

## taskVarAdd()

<b>NAME</b>	<b>taskVarAdd()</b> – add a task variable to a task
<b>SYNOPSIS</b>	<pre><b>STATUS</b> <b>taskVarAdd</b> (     int    tid,                /* ID of task to have new variable */     int * pVar                /* pointer to variable to be switched for task */ )</pre>
<b>DESCRIPTION</b>	<p>This routine adds a specified variable <i>pVar</i> (4-byte memory location) to a specified task's context. After calling this routine, the variable will be private to the task. The task can access and modify the variable, but the modifications will not appear to other tasks, and other tasks' modifications to that variable will not affect the value seen by the task. This is accomplished by saving and restoring the variable's initial value each time a task switch occurs to or from the calling task.</p> <p>This facility can be used when a routine is to be spawned repeatedly as several independent tasks. Although each task will have its own stack, and thus separate stack variables, they will all share the same static and global variables. To make a variable <i>not</i> shareable, the routine can call <b>taskVarAdd()</b> to make a separate copy of the variable for each task, but all at the same physical address.</p>

Note that task variables increase the task switch time to and from the tasks that own them. Therefore, it is desirable to limit the number of task variables that a task uses. One efficient way to use task variables is to have a single task variable that is a pointer to a dynamically allocated structure containing the task's private data.

**EXAMPLE**

Assume that three identical tasks were spawned with a routine called **operator()**. All three use the structure **OP\_GLOBAL** for all variables that are specific to a particular incarnation of the task. The following code fragment shows how this is set up:

```
OP_GLOBAL *opGlobal; /* ptr to operator task's global variables */
void operator
(
  int opNum          /* number of this operator task */
)
{
  if (taskVarAdd (0, (int *)&opGlobal) != OK)
  {
    printErr ("operator%d: can't taskVarAdd opGlobal\n", opNum);
    taskSuspend (0);
  }
  if ((opGlobal = (OP_GLOBAL *) malloc (sizeof (OP_GLOBAL))) == NULL)
  {
    printErr ("operator%d: can't malloc opGlobal\n", opNum);
    taskSuspend (0);
  }
  ...
}
```

**RETURNS** OK, or ERROR if memory is insufficient for the task variable descriptor.

**SEE ALSO** taskVarLib, taskVarDelete(), taskVarGet(), taskVarSet()

---

## taskVarDelete()

<b>NAME</b>	<b>taskVarDelete()</b> – remove a task variable from a task
<b>SYNOPSIS</b>	<pre><b>STATUS</b> taskVarDelete (     int    tid,          /* ID of task whose variable is to be removed */     int * pVar          /* pointer to task variable to be removed */ )</pre>
<b>DESCRIPTION</b>	This routine removes a specified task variable, <i>pVar</i> , from the specified task's context. The private value of that variable is lost.
<b>RETURNS</b>	OK, or ERROR if the task variable does not exist for the specified task.
<b>SEE ALSO</b>	<b>taskVarLib</b> , <b>taskVarAdd()</b> , <b>taskVarGet()</b> , <b>taskVarSet()</b>

---

## taskVarGet()

<b>NAME</b>	<b>taskVarGet()</b> – get the value of a task variable
<b>SYNOPSIS</b>	<pre>int taskVarGet (     int    tid,          /* ID of task whose task variable is to be retrieved */     int * pVar          /* pointer to task variable */ )</pre>
<b>DESCRIPTION</b>	This routine returns the private value of a task variable for a specified task. The specified task is usually not the calling task, which can get its private value by directly accessing the variable. This routine is provided primarily for debugging purposes.
<b>RETURNS</b>	The private value of the task variable, or ERROR if the task is not found or it does not own the task variable.
<b>SEE ALSO</b>	<b>taskVarLib</b> , <b>taskVarAdd()</b> , <b>taskVarDelete()</b> , <b>taskVarSet()</b>

---

## taskVarInfo()

<b>NAME</b>	<b>taskVarInfo()</b> – get a list of task variables of a task
<b>SYNOPSIS</b>	<pre>int taskVarInfo (     int      tid,          /* ID of task whose task variable is to be set */     TASK_VAR varList[],   /* array to hold task variable addresses */     int      maxVars      /* maximum variables varList can accommodate */ )</pre>
<b>DESCRIPTION</b>	<p>This routine provides the calling task with a list of all of the task variables of a specified task. The unsorted array of task variables is copied to <i>varList</i>.</p> <hr/> <p><b>WARNING:</b> Kernel rescheduling is disabled with <b>taskLock()</b> while task variables are looked up. There is no guarantee that all the task variables are still valid or that new task variables have not been created by the time this routine returns.</p> <hr/>
<b>RETURNS</b>	The number of task variables in the list.
<b>SEE ALSO</b>	<b>taskVarLib</b>

---

## taskVarInit()

<b>NAME</b>	<b>taskVarInit()</b> – initialize the task variables facility
<b>SYNOPSIS</b>	<pre>STATUS taskVarInit (void)</pre>
<b>DESCRIPTION</b>	<p>This routine initializes the task variables facility. It installs task switch and delete hooks used for implementing task variables. If <b>taskVarInit()</b> is not called explicitly, <b>taskVarAdd()</b> will call it automatically when the first task variable is added.</p> <p>After the first invocation of this routine, subsequent invocations have no effect.</p> <hr/> <p><b>WARNING:</b> Order dependencies in task delete hooks often involve task variables. If a facility uses task variables and has a task delete hook that expects to use those task variables, the facility's delete hook must run before the task variables' delete hook. Otherwise, the task variables will be deleted by the time the facility's delete hook runs.</p> <hr/>

**taskVarSet()**

VxWorks is careful to run the delete hooks in reverse of the order in which they were installed. Any facility that has a delete hook that will use task variables can guarantee proper ordering by calling **taskVarInit()** before adding its own delete hook.

Note that this is not an issue in normal use of task variables. The issue only arises when adding another task delete hook that uses task variables.

Caution should also be taken when adding task variables from within create hooks. If the task variable package has not been installed via **taskVarInit()**, the create hook attempts to create a create hook, and that may cause system failure. To avoid this situation, **taskVarInit()** should be called during system initialization from the root task, **usrRoot()**, in **usrConfig.c**.

**RETURNS** OK, or ERROR if the task switch/delete hooks could not be installed.

**SEE ALSO** [taskVarLib](#)

---

## taskVarSet()

**NAME** **taskVarSet()** – set the value of a task variable

**SYNOPSIS**

```

STATUS taskVarSet
(
    int    tid,      /* ID of task whose task variable is to be set */
    int * pVar,     /* pointer to task variable to be set for this task */
    int    value    /* new value of task variable */
)

```

**DESCRIPTION** This routine sets the private value of the task variable for a specified task. The specified task is usually not the calling task, which can set its private value by directly modifying the variable. This routine is provided primarily for debugging purposes.

**RETURNS** OK, or ERROR if the task is not found or it does not own the task variable.

**SEE ALSO** [taskVarLib](#), [taskVarAdd\(\)](#), [taskVarDelete\(\)](#), [taskVarGet\(\)](#)

---

## tcpDebugShow()

<b>NAME</b>	<code>tcpDebugShow()</code> – display debugging information for the TCP protocol
<b>SYNOPSIS</b>	<pre>void tcpDebugShow (     int numPrint,          /* no. of entries to print, default (0) = 20 */     int verbose           /* 1 = verbose */ )</pre>
<b>DESCRIPTION</b>	This routine displays debugging information for the TCP protocol. To include TCP debugging facilities, define <code>INCLUDE_TCP_DEBUG</code> when building the system image. To enable information gathering, turn on the <code>SO_DEBUG</code> option for the relevant socket(s).
<b>RETURNS</b>	N/A
<b>SEE ALSO</b>	<code>tcpShow</code>

---

## tcpShowInit()

<b>NAME</b>	<code>tcpShowInit()</code> – initialize TCP show routines
<b>SYNOPSIS</b>	<pre>void tcpShowInit (void)</pre>
<b>DESCRIPTION</b>	This routine links the TCP show facility into the VxWorks system. These routines are included automatically if <code>INCLUDE_TCP_SHOW</code> is defined.
<b>RETURNS</b>	N/A
<b>SEE ALSO</b>	<code>tcpShow</code>

## tcpstatShow()

<b>NAME</b>	<b>tcpstatShow()</b> – display all statistics for the TCP protocol
<b>SYNOPSIS</b>	<code>void tcpstatShow (void)</code>
<b>DESCRIPTION</b>	This routine displays detailed statistics for the TCP protocol.
<b>RETURNS</b>	N/A
<b>SEE ALSO</b>	<b>tcpShow</b>

---

## td()

<b>NAME</b>	<b>td()</b> – delete a task
<b>SYNOPSIS</b>	<pre>void td (     int taskNameOrId      /* task name or task ID */ )</pre>
<b>DESCRIPTION</b>	This command deletes a specified task. It simply calls <b>taskDelete()</b> .
<b>RETURNS</b>	N/A
<b>SEE ALSO</b>	<b>usrLib</b> , <b>taskDelete()</b> , <i>VxWorks Programmer's Guide: Target Shell</i> , <b>windsh</b> , <i>Tornado User's Guide: Shell</i>

---

## telnetdExit()

<b>NAME</b>	<code>telnetdExit()</code> – close an active <b>telnet</b> session
<b>SYNOPSIS</b>	<pre>void telnetdExit (     UINT32 sessionId      /* identifies the session to be deleted */ )</pre>
<b>DESCRIPTION</b>	This routine supports the session exit command for a command interpreter (such as <code>logout()</code> for the VxWorks shell). Depending on the <code>TELNETD_TASKFLAG</code> setting, it causes the associated input and output tasks to restart or exit. <i>sessionId</i> must match a value provided to the command interpreter with the <code>REMOTE_START</code> option.
<b>RETURNS</b>	N/A.
<b>SEE ALSO</b>	<code>telnetdLib</code>

---

## telnetdInit()

<b>NAME</b>	<code>telnetdInit()</code> – initialize the <b>telnet</b> services
<b>SYNOPSIS</b>	<pre>STATUS telnetdInit (     int numClients, /* maximum number of simultaneous sessions */     BOOL staticFlag /* TRUE: create all tasks in advance of any clients */ )</pre>
<b>DESCRIPTION</b>	This routine initializes the <b>telnet</b> server, which supports remote login to VxWorks via the <b>telnet</b> protocol. It is called automatically when the configuration macro <code>INCLUDE_TELNET</code> is defined. The <b>telnet</b> server supports simultaneous client sessions up to the limit specified by the <code>TELNETD_MAX_CLIENTS</code> setting provided in the <i>numClients</i> argument. The <i>staticFlag</i> argument is equal to the <code>TELNETD_TASKFLAG</code> setting. It allows the server to create all of the secondary input and output tasks and allocate all required resources in advance of any connection. The default value of <code>FALSE</code> causes the server to spawn a task pair and create the associated data structures after each new connection.
<b>RETURNS</b>	OK, or <code>ERROR</code> if initialization fails
<b>SEE ALSO</b>	<code>telnetdLib</code>

---

## telnetdParserSet()

**NAME** telnetdParserSet() – specify a command interpreter for **telnet** sessions

**SYNOPSIS**

```
STATUS telnetdParserSet
(
    FUNCPTR pParserCtrlRtn    /* provides parser's file descriptors */
)
```

**DESCRIPTION** This routine provides the ability to handle **telnet** connections using a custom command interpreter or the default VxWorks shell. It is called automatically during system startup (when the configuration macro **INCLUDE\_TELNET** is defined) to connect clients to the command interpreter specified in the **TELNETD\_PARSER\_HOOK** parameter. The command interpreter in use when the **telnet** server start scan never be changed.

The *pParserCtrlRtn* argument provides a routine using the following interface:

```
STATUS parserControlRtn
(
    int telnetdEvent, /* start or stop a telnet session */
    UINT32 sessionId, /* a unique session identifier */
    int ioFd          /* file descriptor for character i/o */
)
```

The **telnet** server calls the control routine with a *telnetdEvent* parameter of **REMOTE\_INIT** during initialization. The **telnet** server then calls the control routine with a *telnetdEvent* parameter of **REMOTE\_START** when a client establishes a new connection. The *sessionId* parameter provides a unique identifier for the session.

In the default configuration, the **telnet** server calls the control routine with a *telnetdEvent* parameter of **REMOTE\_STOP** when a session ends.

The **telnet** server does not call the control routine when a session ends if it is configured to spawn all tasks and allocate all resources in advance of any connections. The associated file descriptors will be reused by later clients and cannot be released. In that case, the **REMOTE\_STOP** operation only occurs to allow the command interpreter to close those files when the server encounters a fatal error.

**RETURNS** OK if parser control routine installed, or **ERROR** otherwise.

**SEE ALSO** telnetdLib

---

## telnetdStart()

**NAME** `telnetdStart()` – initialize the **telnet** services

**SYNOPSIS**

```
STATUS telnetdStart  
(  
    int port                /* target port for accepting connections */  
)
```

**DESCRIPTION** Following the **telnet** server initialization, this routine creates a socket for accepting remote connections and spawns the primary **telnet** server task. It executes automatically during system startup when the **INCLUDE\_TELNET** configuration macro is defined since a parser control routine is available. The server will not accept connections otherwise.

By default, the server will spawn a pair of secondary input and output tasks after each client connection. Changing the **TELNETD\_TASKFLAG** setting to **TRUE** causes this routine to create all of those tasks in advance of any connection. In that case, it calls the current parser control routine repeatedly to obtain file descriptors for each possible client based on the *numClients* argument to the initialization routine. The server will not start if the parser control routine returns **ERROR**.

The **TELNETD\_PORT** constant provides the *port* argument, which assigns the port where the server accepts connections. The default value is the standard setting of 23.

### **VXWORKS AE PROTECTION DOMAINS**

Under VxWorks AE, you can call this function from within the kernel protection domain only. This restriction does not apply under non-AE versions of VxWorks.

**RETURNS** **OK**, or **ERROR** if startup fails

**SEE ALSO** `telnetdLib`

---

## telnetdStaticTaskInitializationGet()

<b>NAME</b>	<code>telnetdStaticTaskInitializationGet()</code> – report whether tasks were pre-started by <code>telnetd</code>
<b>SYNOPSIS</b>	<code>BOOL telnetdStaticTaskInitializationGet ()</code>
<b>DESCRIPTION</b>	This routine is called by a custom shell parser library to determine whether a shell is to be spawned at the time a connection is requested.
<b>RETURNS</b>	<code>TRUE</code> , if all tasks are pre-spawned; <code>FALSE</code> , if tasks are spawned at the time a connection is requested.
<b>SEE ALSO</b>	<code>telnetdLib</code> , <code>telnetdInit()</code> , <code>telnetdParserSet()</code>

---

## tffsBootImagePut()

<b>NAME</b>	<code>tffsBootImagePut()</code> – write to the boot-image region of the flash device
<b>SYNOPSIS</b>	<pre>STATUS tffsBootImagePut (     int    driveNo,          /* TFFS drive number */     int    offset,          /* offset in the flash chip/card */     char * filename         /* binary format of the bootimage */ )</pre>
<b>DESCRIPTION</b>	<p>This routine writes an input stream to the boot-image region (if any) of a flash memory device. Typically, the input stream contains a boot image, such as the VxWorks boot image, but you are free to use this function to write any data needed. The size of the boot-image region is set by the <code>tffsDevFormat()</code> call (or the <code>sysTffsFormat()</code> call, a BSP-specific helper function that calls <code>tffsDevFormat()</code> internally) that formats the flash device for use with TrueFFS.</p> <p>If <code>tffsBootImagePut()</code> is used to put a VxWorks boot image in flash, you should not use the s-record version of the boot image typically produced by <code>make</code>. Instead, you should take the pre s-record version (usually called <code>bootrom</code> instead of <code>bootrom.hex</code>), and filter out its loader header information using an <code>xxxToBin</code> utility. For example:</p> <pre>elfToBin &lt; bootrom &gt; bootrom.bin</pre> <p>Use the resulting <code>bootrom.bin</code> as input to <code>tffsBootImagePut()</code>.</p>

The discussion above assumes that you want only to use the flash device to store a VxWorks image that is retrieved from the flash device and then run out of RAM. However, because it is possible to map many flash devices directly into the target's memory, it is also possible run the VxWorks image from flash memory, although there are some restrictions:

- The flash device must be non-NAND.
- Only the text segment of the VxWorks image (**vxWorks.res\_rom**) may run out of flash memory. The data segment of the image must reside in standard RAM.
- No part of the flash device may be erased while the VxWorks image is running from flash memory.

Because TrueFFS garbage collection triggers an erase, this last restriction means that you cannot run a VxWorks boot image out of a flash device that must also support a writable file system (although a read-only file system is **OK**).

This last restriction arises from the way in which flash devices are constructed. The current physical construction of flash memory devices does not allow access to the device while an erase is in progress anywhere on the flash device. As a result, if TrueFFS tries to erase a portion of the flash device, the entire device becomes inaccessible to all other users. If that other user happens to be the VxWorks image looking for its next instruction, the VxWorks image crashes.

**RETURNS**            **OK** or **ERROR**

**SEE ALSO**            **tffsConfig**

---

## tffsDevCreate( )

**NAME**                **tffsDevCreate( )** – create a TrueFFS block device suitable for use with dosFs

**SYNOPSIS**           **BLK\_DEV \* tffsDevCreate**  
                   (  
                   **int tffsDriveNo,**            **/\* TFFS drive number (0 - DRIVES-1) \*/**  
                   **int removableMediaFlag**    **/\* 0 - nonremovable flash media \*/**  
                   )

**DESCRIPTION**        This routine creates a TFFS block device on top of a flash device. It takes as arguments a drive number, determined from the order in which the socket components were registered, and a flag integer that indicates whether the medium is removable or not. A zero indicates a non removable medium. A one indicates a removable medium. If you

intend to mount dosFs on this block device, you probably do not want to call **tffsDevCreate()**, but should call **usrTffsConfig()** instead. Internally, **usrTffsConfig()** calls **tffsDevCreate()** for you. It then does everything necessary (such as calling the **dosFsDevInit()** routine) to mount dosFs on the just created block device.

**RETURNS** BLK\_DEV pointer, or NULL if it failed.

**SEE ALSO** **tffsDrv**

---

## tffsDevFormat()

**NAME** **tffsDevFormat()** – format a flash device for use with TrueFFS

**SYNOPSIS**

```
STATUS tffsDevFormat
(
    int tffsDriveNo,          /* TrueFFS drive number (0 - DRIVES-1) */
    int arg                  /* pointer to tffsDevFormatParams structure */
)
```

**DESCRIPTION** This routine formats a flash device for use with TrueFFS. It takes two parameters, a drive number and a pointer to a device format structure. This structure describes how the volume should be formatted. The structure is defined in **dosformat.h**. The drive number is assigned in the order that the socket component for the device was registered.

The format process marks each erase unit with an Erase Unit Header (EUH) and creates the physical and virtual Block Allocation Maps (BAM) for the device. The erase units reserved for the “boot-image” are skipped and the first EUH is placed at number (boot-image length - 1). To write to the boot-image region, call **tffsBootImagePut()**.

---

**WARNING:** If any of the erase units in the boot-image region contains an erase unit header from a previous format call (this can happen if you reformat a flash device specifying a larger boot region) TrueFFS fails to mount the device. To fix this problem, use **tffsRawio()** to erase the problem erase units (thus removing the outdated EUH).

---

The macro **TFFS\_STD\_FORMAT\_PARAMS** defines the default values used for formatting a flask disk device. If the second argument to this routine is zero, **tffsDevFormat()** uses these default values.

**RETURNS** OK, or ERROR if it failed.

**SEE ALSO** **tffsDrv**

---

## tffsDevOptionsSet()

<b>NAME</b>	<b>tffsDevOptionsSet()</b> – set TrueFFS volume options
<b>SYNOPSIS</b>	<pre><b>STATUS</b> tffsDevOptionsSet (     <b>TFFS_DEV</b> * pTffsDev      /* pointer to device descriptor */ )</pre>
<b>DESCRIPTION</b>	This routine is intended to set various TrueFFS volume options. At present it only disables FAT monitoring. If VxWorks long file names are to be used with TrueFFS, FAT monitoring must be turned off.
<b>RETURNS</b>	OK, or ERROR if it failed.
<b>SEE ALSO</b>	tffsDrv

---

## tffsDrv()

<b>NAME</b>	<b>tffsDrv()</b> – initialize the TrueFFS system
<b>SYNOPSIS</b>	<pre><b>STATUS</b> tffsDrv (void)</pre>
<b>DESCRIPTION</b>	<p>This routine sets up the structures, the global variables, and the mutual exclusion semaphore needed to manage TrueFFS. This call also registers socket component drivers for all the flash devices attached to your target.</p> <p>Because <b>tffsDrv()</b> is the call that initializes the TrueFFS system, this function must be called (exactly once) before calling any other TrueFFS utilities, such as <b>tffsDevFormat()</b> or <b>tffsDevCreate()</b>. Typically, the call to <b>tffsDrv()</b> is handled for you automatically. If you defined <b>INCLUDE_TFFS</b> in your BSP's <b>config.h</b>, the call to <b>tffsDrv()</b> is made from <b>usrRoot()</b>. If your BSP's <b>config.h</b> defines <b>INCLUDE_PCMCIA</b>, the call to <b>tffsDrv()</b> is made from <b>pccardTffsEnabler()</b>.</p>
<b>RETURNS</b>	OK, or ERROR if it fails.
<b>SEE ALSO</b>	tffsDrv

---

## tffsRawio()

**NAME** tffsRawio() – low level I/O access to flash components

**SYNOPSIS**

```
STATUS tffsRawio
(
    int tffsDriveNo,          /* TrueFFS drive number (0 - DRIVES-1) */
    int functionNo,          /* TrueFFS function code */
    int arg0,                /* argument 0 */
    int arg1,                /* argument 1 */
    int arg2                 /* argument 2 */
)
```

**DESCRIPTION** Use the utilities provided by this routine with the utmost care. If you use these routines carelessly, you risk data loss as well as permanent physical damage to the flash device.

This routine is a gateway to a series of utilities (listed below). Functions such as **mkbootTffs()** and **tffsBootImagePut()** use these **tffsRawio()** utilities to write boot sector information. The functions for physical read, write, and erase are made available with the intention that they be used on erase units allocated to the boot-image region by **tffsDevFormat()**. Using these functions elsewhere could be dangerous.

The *arg0*, *arg1*, and *arg2* parameters to **tffsRawio()** are interpreted differently depending on the function number you specify for *functionNo*. The drive number is determined by the order in which the socket components were registered.

Function Name	arg0	arg1	arg2
TFFS_GET_PHYSICAL_INFO	user buffer address	N/A	N/A
TFFS_PHYSICAL_READ	address to read	byte count	user buffer address
TFFS_PHYSICAL_WRITE	address to write	byte count	user buffer address
TFFS_PHYSICAL_ERASE	first unit	number of units	N/A
TFFS_ABS_READ	sector number	number of sectors	user buffer address
TFFS_ABS_WRITE	sector number	number of sectors	user buffer address
TFFS_ABS_DELETE	sector number	number of sectors	N/A
TFFS_DEFRAGMENT_VOLUME	number of sectors	user buffer address	N/A

**TFFS\_GET\_PHYSICAL\_INFO** writes the flash type, erasable block size, and media size to the user buffer specified in *arg0*.

**TFFS\_PHYSICAL\_READ** reads *arg1* bytes from *arg0* and writes them to the buffer specified by *arg2*.

**TFFS\_PHYSICAL\_WRITE** copies *arg1* bytes from the *arg2* buffer and writes them to the flash memory location specified by *arg0*. This aborts if the volume is already mounted to

prevent the versions of translation data in memory and in flash from going out of synchronization.

**TFFS\_PHYSICAL\_ERASE** erases *arg1* erase units, starting at the erase unit specified in *arg0*. This aborts if the volume is already mounted to prevent the versions of translation data in memory and in flash from going out of synchronization.

**TFFS\_ABS\_READ** reads *arg1* sectors, starting at sector *arg0*, and writes them to the user buffer specified in *arg2*.

**TFFS\_ABS\_WRITE** takes data from the *arg2* user buffer and writes *arg1* sectors of it to the flash location starting at sector *arg0*.

**TFFS\_ABS\_DELETE** deletes *arg1* sectors of data starting at sector *arg0*.

**TFFS\_DEFRAGMENT\_VOLUME** calls the defragmentation routine with the minimum number of sectors to be reclaimed, *arg0*, and writes the actual number reclaimed in the user buffer by *arg1*. Calling this function through some low priority task will make writes more deterministic. No validation is done of the user specified address fields, so the functions assume they are writable. If the address is invalid, you could see bus errors or segmentation faults.

**RETURNS** OK, or **ERROR** if it failed.

**SEE ALSO** `tffsDrv`

---

## tffsShow()

**NAME** `tffsShow()` – show device information on a specific socket interface

**SYNOPSIS**

```
void tffsShow
(
    int driveNo          /* TFFS drive number */
)
```

**DESCRIPTION** This routine prints device information on the specified socket interface. This information is particularly useful when trying to determine the number of Erase Units required to contain a boot image. The field called *unitSize* reports the size of an Erase Unit.

If the process of getting physical information fails, an error code is printed. The error codes can be found in **flbase.h**.

**RETURNS** N/A

**SEE ALSO** `tffsConfig`

---

## tffsShowAll()

<b>NAME</b>	<b>tffsShowAll()</b> – show device information on all socket interfaces
<b>SYNOPSIS</b>	<code>void tffsShowAll (void)</code>
<b>DESCRIPTION</b>	This routine prints device information on all socket interfaces.
<b>RETURNS</b>	N/A
<b>SEE ALSO</b>	<b>tffsConfig</b>

---

## tftpCopy()

<b>NAME</b>	<b>tftpCopy()</b> – transfer a file via TFTP
<b>SYNOPSIS</b>	<pre><b>STATUS</b> tftpCopy (     char * pHost,           /* host name or address */     int   port,            /* optional port number */     char * pFilename,      /* remote filename */     char * pCommand,       /* TFTP command */     char * pMode,          /* TFTP transfer mode */     int   fd               /* fd to put/get data */ )</pre>

**DESCRIPTION** This routine transfers a file using the TFTP protocol to or from a remote system. *pHost* is the remote server name or Internet address. A non-zero value for *port* specifies an alternate TFTP server port (zero means use default TFTP port number (69)). *pFilename* is the remote file name. *pCommand* specifies the TFTP command, which can be either “put” or “get”. *pMode* specifies the mode of transfer, which can be “ascii”, “netascii”, “binary”, “image”, or “octet”.

*fd* is a file descriptor from which to read/write the data from or to the remote system. For example, if the command is “get”, the remote data will be written to *fd*. If the command is “put”, the data to be sent is read from *fd*. The caller is responsible for managing *fd*. That is, *fd* must be opened prior to calling **tftpCopy()** and closed up on completion.

**EXAMPLE** The following sequence gets an ASCII file `/folk/vw/xx.yy` on host “congo” and stores it to a local file called `localfile`:

```
-> fd = open ("localfile", 0x201, 0644)
-> tftpdCopy ("congo", 0, "/folk/vw/xx.yy", "get", "ascii", fd)
-> close (fd)
```

**RETURNS** OK, or ERROR if unsuccessful.

**ERRNO** S\_tftplib\_INVALID\_COMMAND

**SEE ALSO** tftplib, ftpLib

---

## tftpdDirectoryAdd()

**NAME** tftpdDirectoryAdd() – add a directory to the access list

**SYNOPSIS**

```
STATUS tftpdDirectoryAdd
(
    char * fileName          /* name of directory to add to access list */
)
```

**DESCRIPTION** This routine adds the specified directory name to the access list for the TFTP server.

**RETURNS** N/A

**SEE ALSO** tftplib

---

## tftpdDirectoryRemove()

**NAME** tftpdDirectoryRemove() – delete a directory from the access list

**SYNOPSIS**

```
STATUS tftpdDirectoryRemove
(
    char * fileName          /* name of directory to add to access list */
)
```

**DESCRIPTION** This routine deletes the specified directory name from the access list for the TFTP server.

**RETURNS** N/A

**SEE ALSO** **tftpdLib**

---

## tftpdInit()

**NAME** **tftpdInit()** – initialize the TFTP server task

**SYNOPSIS**

```
STATUS tftpdInit
(
    int    stackSize,          /* stack size for the tftpdTask */
    int    nDirectories,      /* number of directories allowed read */
    char * *directoryNames,   /* array of dir names */
    BOOL   noControl,         /* TRUE if no access control required */
    int    maxConnections
)
```

**DESCRIPTION** This routine will spawn a new TFTP server task, if one does not already exist. If a TFTP server task is running already, **tftpdInit()** will simply return an **ERROR** value without creating a new task.

To change the default stack size for the TFTP server task, use the *stackSize* parameter. The task stack size should be set to a large enough value for the needs of your application - use **checkStack()** to evaluate your stack usage. The default size is set in the global variable **tftpdTaskStackSize**. Setting *stackSize* to zero will result in the stack size being set to this default.

To set the maximum number of simultaneous TFTP connections (each with its own transfer identifier or TID), set the *maxConnections* parameter. More information on this is found in RFC 1350 ("The TFTP Protocol (Revision 2)"). Setting *maxConnections* to zero will result in the maximum number of connections being set to the default, which is 10.

If *noControl* is **TRUE**, the server will be set up to transfer any file in any location. Otherwise, it will only transfer files in the directories in **/tftpboot** or the *nDirectories* directories in the *directoryNames* list, and will send an access violation error to clients that attempt to access files outside of these directories.

By default, *noControl* is **FALSE**, *directoryNames* is empty, *nDirectories* is zero, and access is restricted to the **/tftpboot** directory.

Directories can be added to the access list after initialization by using the **tftpdDirectoryAdd()** routine.

**VXWORKS AE PROTECTION DOMAINS**

Under VxWorks AE, you can call this function from within the kernel protection domain only. In addition, all arguments to this function can reference only that data which is valid in the kernel protection domain. This restriction does not apply under non-AE versions of VxWorks.

**RETURNS** OK, or **ERROR** if a new TFTP task cannot be created.

**SEE ALSO** **tftpdLib**

---

## tftpdTask()

**NAME** **tftpdTask()** – TFTP server daemon task

**SYNOPSIS**

```
STATUS tftpdTask  
(  
    int    nDirectories,      /* number of dirs allowed access */  
    char * *directoryNames,  /* array of directory names */  
    int    maxConnections    /* max number of simultan. connects */  
)
```

**DESCRIPTION** This routine processes incoming TFTP client requests by spawning a new task for each connection that is set up. This routine is called by **tftpdInit()**.

**VXWORKS AE PROTECTION DOMAINS**

Under VxWorks AE, you can call this function from within the kernel protection domain only. In addition, all arguments to this function can reference only that data which is valid in the kernel protection domain. This restriction does not apply under non-AE versions of VxWorks.

**RETURNS** OK, or **ERROR** if the task returns unexpectedly.

**SEE ALSO** **tftpdLib**

## tftpGet()

**NAME** tftpGet() – get a file from a remote system

**SYNOPSIS**

```
STATUS tftpGet
(
    TFTP_DESC * pTftpDesc,    /* TFTP descriptor */
    char *      pFilename,   /* remote filename */
    int         fd,          /* file descriptor */
    int         clientOrServer /* which side is calling */
)
```

**DESCRIPTION** This routine gets a file from a remote system via TFTP. *pFilename* is the filename. *fd* is the file descriptor to which the data is written. *pTftpDesc* is a pointer to the TFTP descriptor. The **tftpPeerSet()** routine must be called prior to calling this routine.

**RETURNS** OK, or ERROR if unsuccessful.

**ERRNO** S\_tftpLib\_INVALID\_DESCRIPTOR  
S\_tftpLib\_INVALID\_ARGUMENT  
S\_tftpLib\_NOT\_CONNECTED

**SEE ALSO** tftpLib

---

## tftpInfoShow()

**NAME** tftpInfoShow() – get TFTP status information

**SYNOPSIS**

```
STATUS tftpInfoShow
(
    TFTP_DESC * pTftpDesc    /* TFTP descriptor */
)
```

**DESCRIPTION** This routine prints information associated with TFTP descriptor *pTftpDesc*.

**EXAMPLE** A call to **tftpInfoShow()** might look like:

```
-> tftpInfoShow (tftpDesc)
    Connected to yuba [69]
    Mode: netascii Verbose: off Tracing: off
    Rexmt-interval: 5 seconds, Max-timeout: 25 seconds
```

**RETURNS** OK, or **ERROR** if unsuccessful.

**ERRNO** **S\_tftpLib\_INVALID\_DESCRIPTOR**

**SEE ALSO** **tftpLib**

---

## **tftpInit()**

**NAME** **tftpInit()** – initialize a TFTP session

**SYNOPSIS** **TFTP\_DESC \* tftpInit (void)**

**DESCRIPTION** This routine initializes a TFTP session by allocating and initializing a TFTP descriptor. It sets the default transfer mode to “netascii”.

**RETURNS** A pointer to a TFTP descriptor if successful, otherwise **NULL**.

**SEE ALSO** **tftpLib**

---

## **tftpModeSet()**

**NAME** **tftpModeSet()** – set the TFTP transfer mode

**SYNOPSIS** **STATUS tftpModeSet**

```
(  
    TFTP_DESC * pTftpDesc,    /* TFTP descriptor */  
    char *      pMode        /* TFTP transfer mode */  
)
```

**DESCRIPTION** This routine sets the transfer mode associated with the TFTP descriptor *pTftpDesc*. *pMode* specifies the transfer mode, which can be “netascii”, “binary”, “image”, or “octet”. Although recognized, these modes actually translate into either octet or netascii.

**RETURNS** OK, or **ERROR** if unsuccessful.

**ERRNO** **S\_tftpLib\_INVALID\_DESCRIPTOR**, **S\_tftpLib\_INVALID\_ARGUMENT**, **S\_tftpLib\_INVALID\_MODE**

**SEE ALSO** **tftpLib**

## tftpPeerSet()

**NAME** tftpPeerSet() – set the TFTP server address

**SYNOPSIS**

```
STATUS tftpPeerSet
(
    TFTP_DESC * pTftpDesc,    /* TFTP descriptor */
    char *      pHostname,    /* server name/address */
    int         port          /* port number */
)
```

**DESCRIPTION** This routine sets the TFTP server (peer) address associated with the TFTP descriptor *pTftpDesc*. *pHostname* is either the TFTP server name (e.g., “congo”) or the server Internet address (e.g., “90.3”). A non-zero value for *port* specifies the server port number (zero means use the default TFTP server port number (69)).

**RETURNS** OK, or ERROR if unsuccessful.

**ERRNO** S\_tftpLib\_INVALID\_DESCRIPTOR  
S\_tftpLib\_INVALID\_ARGUMENT  
S\_tftpLib\_UNKNOWN\_HOST

**SEE ALSO** tftpLib

---

## tftpPut()

**NAME** tftpPut() – put a file to a remote system

**SYNOPSIS**

```
STATUS tftpPut
(
    TFTP_DESC * pTftpDesc,    /* TFTP descriptor */
    char *      pFilename,    /* remote filename */
    int         fd,          /* file descriptor */
    int         clientOrServer /* which side is calling */
)
```

**DESCRIPTION** This routine puts data from a local file (descriptor) to a file on the remote system. *pTftpDesc* is a pointer to the TFTP descriptor. *pFilename* is the remote filename. *fd* is the file descriptor from which it gets the data. A call to **tftpPeerSet()** must be made prior to calling this routine.

**RETURNS** OK, or **ERROR** if unsuccessful.

**ERRNO** S\_tftpLib\_INVALID\_DESCRIPTOR  
S\_tftpLib\_INVALID\_ARGUMENT  
S\_tftpLib\_NOT\_CONNECTED

**SEE ALSO** tftpLib

## tftpQuit()

**NAME** tftpQuit() – quit a TFTP session

**SYNOPSIS**

```
STATUS tftpQuit
(
    TFTP_DESC * pTftpDesc    /* TFTP descriptor */
)
```

**DESCRIPTION** This routine closes a TFTP session associated with the TFTP descriptor *pTftpDesc*.

**RETURNS** OK, or **ERROR** if unsuccessful.

**ERRNO** S\_tftpLib\_INVALID\_DESCRIPTOR

**SEE ALSO** tftpLib

## tftpSend()

**NAME** tftpSend() – send a TFTP message to the remote system

**SYNOPSIS**

```
int tftpSend
(
    TFTP_DESC * pTftpDesc,    /* TFTP descriptor */
    TFTP_MSG * pTftpMsg,     /* TFTP send message */
    int        sizeMsg,      /* send message size */
    TFTP_MSG * pTftpReply,   /* TFTP reply message */
    int        opReply,      /* reply opcode */
    int        blockReply,   /* reply block number */
    int *      pPort         /* return port number */
)
```

**tftpXfer()**

<b>DESCRIPTION</b>	This routine sends <i>sizeMsg</i> bytes of the passed message <i>pTftpMsg</i> to the remote system associated with the TFTP descriptor <i>pTftpDesc</i> . If <i>pTftpReply</i> is not NULL, <b>tftpSend()</b> tries to get a reply message with a block number <i>blockReply</i> and an opcode <i>opReply</i> . If <i>pPort</i> is NULL, the reply message must come from the same port to which the message was sent. If <i>pPort</i> is not NULL, the port number from which the reply message comes is copied to this variable.
<b>RETURNS</b>	The size of the reply message, or <b>ERROR</b> .
<b>ERRNO</b>	<b>S_tftpLib_TIMED_OUT</b> <b>S_tftpLib_TFTP_ERROR</b>
<b>SEE ALSO</b>	<b>tftpLib</b>

---

## tftpXfer()

**NAME** **tftpXfer()** – transfer a file via TFTP using a stream interface

**SYNOPSIS**

```

STATUS tftpXfer
(
    char * pHost,           /* host name or address */
    int   port,            /* port number */
    char * pFilename,      /* remote filename */
    char * pCommand,       /* TFTP command */
    char * pMode,          /* TFTP transfer mode */
    int *  pDataDesc,      /* return data desc. */
    int *  pErrorDesc      /* return error desc. */
)

```

**DESCRIPTION** This routine initiates a transfer to or from a remote file via TFTP. It spawns a task to perform the TFTP transfer and returns a descriptor from which the data can be read (for “get”) or to which it can be written (for “put”) interactively. The interface for this routine is similar to **ftpXfer()** in **ftpLib**.

*pHost* is the server name or Internet address. A non-zero value for *port* specifies an alternate TFTP server port number (zero means use default TFTP port number (69)). *pFilename* is the remote filename. *pCommand* specifies the TFTP command. The command can be either “put” or “get”.

The **tftpXfer()** routine returns a data descriptor, in *pDataDesc*, from which the TFTP data is read (for “get”) or to which it is written (for “put”). An error status descriptor is returned in the variable *pErrorDesc*. If an error occurs during the TFTP transfer, an error

string can be read from this descriptor. After returning successfully from **tftpXfer()**, the calling application is responsible for closing both descriptors.

If there are delays in reading or writing the data descriptor, it is possible for the TFTP transfer to time out.

**EXAMPLE**

The following code demonstrates how **tftpXfer()** may be used:

```
#include "tftpLib.h"
#define BUFFERSIZE      512
int  dataFd;
int  errorFd;
int  num;
char buf [BUFFERSIZE + 1];
if (tftpXfer ("congo", 0, "/usr/fred", "get", "ascii", &dataFd,
             &errorFd) == ERROR)
    return (ERROR);
while ((num = read (dataFd, buf, sizeof (buf))) > 0)
    {
        ....
    }
close (dataFd);
num = read (errorFd, buf, BUFFERSIZE);
if (num > 0)
    {
        buf [num] = '\0';
        printf ("YIKES! An error occurred!:%s\n", buf);
        .....
    }
close (errorFd);
```

**RETURNS**

OK, or ERROR if unsuccessful.

**ERRNO**

S\_tftpLib\_INVALID\_ARGUMENT

**SEE ALSO**

tftpLib, ftpLib

**ti()****ti()**

**NAME** `ti()` – print complete information from a task’s TCB

**SYNOPSIS**

```
void ti
(
    int taskNameOrId      /* task name or task ID; 0 = use default */
)
```

**DESCRIPTION** This command prints the task control block (TCB) contents, including registers, for a specified task. If *taskNameOrId* is omitted or zero, the last task referenced is assumed.

The `ti()` routine uses `taskShow()`; see the documentation for `taskShow()` for a description of the output format.

**EXAMPLE** The following shows the TCB contents for the shell task:

```
-> ti
  NAME      ENTRY      TID  PRI  STATUS      PC      SP      ERRNO  DELAY
-----
tShell     _shell      20efcac  1  READY      201dc90  20ef980      0      0
stack: base 0x20efcac  end 0x20ed59c  size 9532  high 1452  margin 8080
options: 0x1e
VX_UNBREAKABLE      VX_DEALLOC_STACK      VX_FP_TASK      VX_STDIO
D0 =      0  D4 =      0  A0 =      0  A4 =      0
D1 =      0  D5 =      0  A1 =      0  A5 = 203a084  SR =      3000
D2 =      0  D6 =      0  A2 =      0  A6 = 20ef9a0  PC = 2038614
D3 =      0  D7 =      0  A3 =      0  A7 = 20ef980
value = 34536868 = 0x20efda4
```

**RETURNS** N/A

**SEE ALSO** `usrLib`, `taskShow()`, *VxWorks Programmer’s Guide: Target Shell*, `windsh`, *Tornado User’s Guide: Shell*

---

## tickAnnounce()

<b>NAME</b>	<b>tickAnnounce()</b> – announce a clock tick to the kernel
<b>SYNOPSIS</b>	<b>void tickAnnounce (void)</b>
<b>DESCRIPTION</b>	This routine informs the kernel of the passing of time. It should be called from an interrupt service routine that is connected to the system clock. The most common frequencies are 60Hz or 100Hz. Frequencies in excess of 600Hz are an inefficient use of processor power because the system will spend most of its time advancing the clock. By default, this routine is called by <b>usrClock()</b> in <b>usrConfig.c</b> .
<b>RETURNS</b>	N/A
<b>SEE ALSO</b>	<b>tickLib, kernelLib, taskLib, semLib, wdLib</b> , <i>VxWorks Programmer's Guide: Basic OS</i>

---

## tickGet()

<b>NAME</b>	<b>tickGet()</b> – get the value of the kernel's tick counter
<b>SYNOPSIS</b>	<b>ULONG tickGet (void)</b>
<b>DESCRIPTION</b>	This routine returns the current value of the tick counter. This value is set to zero at startup, incremented by <b>tickAnnounce()</b> , and can be changed using <b>tickSet()</b> .
<b>RETURNS</b>	The most recent <b>tickSet()</b> value, plus all <b>tickAnnounce()</b> calls since.
<b>SEE ALSO</b>	<b>tickLib, tickSet(), tickAnnounce()</b>

**tickSet()**

---

**tickSet()****NAME** tickSet() – set the value of the kernel's tick counter**SYNOPSIS**  

```
void tickSet
(
    ULONG ticks          /* new time in ticks */
)
```

**DESCRIPTION** This routine sets the internal tick counter to a specified value in ticks. The new count will be reflected by **tickGet()**, but will not change any delay fields or timeouts selected for any tasks. For example, if a task is delayed for ten ticks, and this routine is called to advance time, the delayed task will still be delayed until ten **tickAnnounce()** calls have been made.**RETURNS** N/A**SEE ALSO** tickLib, tickGet(), tickAnnounce()

---

**time()****NAME** time() – determine the current calendar time (ANSI)**SYNOPSIS**  

```
time_t time
(
    time_t * timer      /* calendar time in seconds */
)
```

**DESCRIPTION** This routine returns the implementation's best approximation of current calendar time in seconds. If *timer* is non-NULL, the return value is also copied to the location to which *timer* points.**INCLUDE FILES** time.h**RETURNS** The current calendar time in seconds, or **ERROR** (-1) if the calendar time is not available.**SEE ALSO** ansiTime, clock\_gettime()

---

## timer\_cancel()

**NAME** `timer_cancel()` – cancel a timer

**SYNOPSIS**

```
int timer_cancel
(
    timer_t timerid          /* timer ID */
)
```

**DESCRIPTION** This routine is a shorthand method of invoking `timer_settime()`, which stops a timer.

---

**NOTE:** Non-POSIX.

---

**RETURNS** 0 (OK), or -1 (ERROR) if *timerid* is invalid.

**ERRNO** EINVAL

**SEE ALSO** `timerLib`

---

## timer\_connect()

**NAME** `timer_connect()` – connect a user routine to the timer signal

**SYNOPSIS**

```
int timer_connect
(
    timer_t timerid,        /* timer ID */
    VOIDFUNCPTR routine,    /* user routine */
    int arg                 /* user argument */
)
```

**DESCRIPTION** This routine sets the specified *routine* to be invoked with *arg* when fielding a signal indicated by the timer's *evp* signal number, or if *evp* is NULL, when fielding the default signal (SIGALRM).

The signal handling routine should be declared as:

```
void my_handler
(
    timer_t timerid,        /* expired timer ID */
    int arg                 /* user argument */
)
```

**timer\_create()**


---

**NOTE:** Non-POSIX.

---

**RETURNS** 0 (OK), or -1 (ERROR) if the timer is invalid or cannot bind the signal handler.

**ERRNO** EINVAL

**SEE ALSO** timerLib

---

## timer\_create()

**NAME** timer\_create() – allocate a timer using the specified clock for a timing base (POSIX)

**SYNOPSIS**

```
int timer_create
(
    clockid_t      clock_id, /* clock ID (always CLOCK_REALTIME) */
    struct sigevent * evp,   /* user event handler */
    timer_t *      pTimer   /* ptr to return value */
)
```

**DESCRIPTION** This routine returns a value in *pTimer* that identifies the timer in subsequent timer requests. The *evp* argument, if non-NULL, points to a **sigevent** structure, which is allocated by the application and defines the signal number and application-specific data to be sent to the task when the timer expires. If *evp* is NULL, a default signal (SIGALRM) is queued to the task, and the signal data is set to the timer ID. Initially, the timer is disarmed.

**RETURNS** 0 (OK), or -1 (ERROR) if too many timers already are allocated or the signal number is invalid.

**ERRNO** EMTIMERS, EINVAL, ENOSYS, EAGAIN, S\_memLib\_NOT\_ENOUGH\_MEMORY

**SEE ALSO** timerLib, timer\_delete()

---

## timer\_delete()

**NAME** `timer_delete()` – remove a previously created timer (POSIX)

**SYNOPSIS**

```
int timer_delete
(
    timer_t timerid          /* timer ID */
)
```

**DESCRIPTION** This routine removes a timer.

**RETURNS** 0 (OK), or -1 (ERROR) if *timerid* is invalid.

**ERRNO** EINVAL

**SEE ALSO** `timerLib`, `timer_create()`

---

## timer\_getoverrun()

**NAME** `timer_getoverrun()` – return the timer expiration overrun (POSIX)

**SYNOPSIS**

```
int timer_getoverrun
(
    timer_t timerid          /* timer ID */
)
```

**DESCRIPTION** This routine returns the timer expiration overrun count for *timerid*, when called from a timer expiration signal catcher. The overrun count is the number of extra timer expirations that have occurred, up to the implementation-defined maximum `_POSIX_DELAYTIMER_MAX`. If the count is greater than the maximum, it returns the maximum.

**RETURNS** The number of overruns, or `_POSIX_DELAYTIMER_MAX` if the count equals or is greater than `_POSIX_DELAYTIMER_MAX`, or -1 (ERROR) if *timerid* is invalid.

**ERRNO** EINVAL, ENOSYS

**SEE ALSO** `timerLib`

---

## timer\_gettime()

**NAME** timer\_gettime() – get the remaining time before expiration and the reload value (POSIX)

**SYNOPSIS**

```
int timer_gettime
(
    timer_t          timerid, /* timer ID */
    struct itimerspec * value  /* where to return remaining time */
)
```

**DESCRIPTION** This routine gets the remaining time and reload value of a specified timer. Both values are copied to the *value* structure.

**RETURNS** 0 (OK), or -1 (ERROR) if *timerid* is invalid.

**ERRNO** EINVAL

**SEE ALSO** timerLib

---

## timer\_settime()

**NAME** timer\_settime() – set the time until the next expiration and arm timer (POSIX)

**SYNOPSIS**

```
int timer_settime
(
    timer_t          timerid, /* timer ID */
    int              flags,   /* absolute or relative */
    const struct itimerspec * value, /* time to be set */
    struct itimerspec *  ovalue /* previous time set (NULL=no result) */
)
```

**DESCRIPTION** This routine sets the next expiration of the timer, using the *.it\_value* of *value*, thus arming the timer. If the timer is already armed, this call resets the time until the next expiration. If *.it\_value* is zero, the timer is disarmed.

If *flags* is not equal to **TIMER\_ABSTIME**, the interval is relative to the current time, the interval being the *.it\_value* of the *value* parameter. If *flags* is equal to **TIMER\_ABSTIME**, the expiration is set to the difference between the absolute time of *.it\_value* and the current value of the clock associated with *timerid*. If the time has already passed, then the timer expiration notification is made immediately. The task that sets the timer receives the

signal; in other words, the *taskId* is noted. If a timer is set by an ISR, the signal is delivered to the task that created the timer.

The reload value of the timer is set to the value specified by the `.it_interval` field of *value*. When a timer is armed with a nonzero `.it_interval` a periodic timer is set up.

Time values that are between two consecutive non-negative integer multiples of the resolution of the specified timer are rounded up to the larger multiple of the resolution.

If *ovalue* is non-NULL, the routine stores a value representing the previous amount of time before the timer would have expired. Or if the timer is disarmed, the routine stores zero, together with the previous timer reload value. The *ovalue* parameter is the same value as that returned by `timer_gettime()` and is subject to the timer resolution.

---

**WARNING:** If `clock_settime()` is called to reset the absolute clock time after a timer has been set with `timer_settime()`, and if *flags* is equal to `TIMER_ABSTIME`, then the timer will behave unpredictably. If you must reset the absolute clock time after setting a timer, do not use *flags* equal to `TIMER_ABSTIME`.

---

<b>RETURNS</b>	0 (OK), or -1 (ERROR) if <i>timerid</i> is invalid, the number of nanoseconds specified by <i>value</i> is less than 0 or greater than or equal to 1,000,000,000, or the time specified by <i>value</i> exceeds the maximum allowed by the timer.
<b>ERRNO</b>	EINVAL
<b>SEE ALSO</b>	timerLib

---

## timex()

**NAME** `timex()` – time a single execution of a function or functions

**SYNOPSIS**

```
void timex
(
    FUNCPTR func,           /* function to time (optional) */
    int arg1,              /* first of up to 8 args to call function */
                           /* with (optional) */
    int arg2,
    int arg3,
    int arg4,
    int arg5,
    int arg6,
    int arg7,
    int arg8
)
```

<b>DESCRIPTION</b>	<p>This routine times a single execution of a specified function with up to eight of the function's arguments. If no function is specified, it times the execution of the current list of functions to be timed, which is created using <b>timexFunc()</b>, <b>timexPre()</b>, and <b>timexPost()</b>. If <b>timex()</b> is executed with a function argument, the entire current list is replaced with the single specified function.</p> <p>When execution is complete, <b>timex()</b> displays the execution time. If the execution was so fast relative to the clock rate that the time is meaningless (error &gt; 50%), a warning message is printed instead. In such cases, use <b>timexN()</b>.</p>
<b>RETURNS</b>	N/A
<b>SEE ALSO</b>	<b>timexLib</b> , <b>timexFunc()</b> , <b>timexPre()</b> , <b>timexPost()</b> , <b>timexN()</b>

---

## timexClear()

<b>NAME</b>	<b>timexClear()</b> – clear the list of function calls to be timed
<b>SYNOPSIS</b>	<pre>void timexClear (void)</pre>
<b>DESCRIPTION</b>	This routine clears the current list of functions to be timed.
<b>RETURNS</b>	N/A
<b>SEE ALSO</b>	<b>timexLib</b>

---

## timexFunc()

<b>NAME</b>	<b>timexFunc()</b> – specify functions to be timed
<b>SYNOPSIS</b>	<pre>void timexFunc (     int    i,           /* function number in list (0..3) */     FUNCPTR func,      /* function to be added (NULL if to be deleted) */     int    arg1,       /* first of up to 8 args to call function with */     int    arg2,     int    arg3,     int    arg4,</pre>

```

    int    arg5,
    int    arg6,
    int    arg7,
    int    arg8
)

```

- DESCRIPTION** This routine adds or deletes functions in the list of functions to be timed as a group by calls to **timex()** or **timexN()**. Up to four functions can be included in the list. The argument *i* specifies the function's position in the sequence of execution (0, 1, 2, or 3). A function is deleted by specifying its sequence number *i* and NULL for the function argument *func*.
- RETURNS** N/A
- SEE ALSO** **timexLib**, **timex()**, **timexN()**

---

## timexHelp()

- NAME** **timexHelp()** – display synopsis of execution timer facilities
- SYNOPSIS** **void timexHelp (void)**
- DESCRIPTION** This routine displays the following summary of the available execution timer functions:

<b>timexHelp</b>		Print this list.
<b>timex</b>	[ <b>func</b> , [ <b>args...</b> ]]	Time a single execution.
<b>timexN</b>	[ <b>func</b> , [ <b>args...</b> ]]	Time repeated executions.
<b>timexClear</b>		Clear all functions.
<b>timexFunc</b>	<b>i</b> , <b>func</b> , [ <b>args...</b> ]	Add timed function number <b>i</b> (0,1,2,3).
<b>timexPre</b>	<b>i</b> , <b>func</b> , [ <b>args...</b> ]	Add pre-timing function number <b>i</b> .
<b>timexPost</b>	<b>i</b> , <b>func</b> , [ <b>args...</b> ]	Add post-timing function number <b>i</b> .
<b>timexShow</b>		Show all functions to be called.

**Notes:**

- 1) **timexN()** will repeat calls enough times to get timing accuracy to approximately 2%.
- 2) A single function can be specified with **timex()** and **timexN()**; or, multiple functions can be pre-set with **timexFunc()**.
- 3) Up to 4 functions can be pre-set with **timexFunc()**, **timexPre()**, and **timexPost()**, i.e., **i** in the range 0 - 3.
- 4) **timexPre()** and **timexPost()** allow locking/unlocking, or raising/lowering priority before/after timing.

**timexInit()**

**RETURNS** N/A

**SEE ALSO** [timexLib](#)

**timexInit()**

**NAME** **timexInit()** – include the execution timer library

**SYNOPSIS** `void timexInit (void)`

**DESCRIPTION** This null routine is provided so that **timexLib** can be linked into the system. If the configuration macro `INCLUDE_TIMEX` is defined, it is called by the root task, **usrRoot()**, in **usrConfig.c**.

**RETURNS** N/A

**SEE ALSO** [timexLib](#)

**timexN()**

**NAME** **timexN()** – time repeated executions of a function or group of functions

**SYNOPSIS**

```
void timexN
(
    FUNCPTR func,          /* function to time (optional) */
    int     arg1,          /* first of up to 8 args to call function with */
    int     arg2,
    int     arg3,
    int     arg4,
    int     arg5,
    int     arg6,
    int     arg7,
    int     arg8
)
```

**DESCRIPTION** This routine times the execution of the current list of functions to be timed in the same manner as **timex()**; however, the list of functions is called a variable number of times until

sufficient resolution is achieved to establish the time with an error less than 2%. (Since each iteration of the list may be measured to a resolution of +/- 1 clock tick, repetitive timings decrease this error to 1/N ticks, where N is the number of repetitions.)

**RETURNS** N/A

**SEE ALSO** `timexLib`, `timexFunc()`, `timex()`

---

## timexPost()

**NAME** `timexPost()` – specify functions to be called after timing

**SYNOPSIS**

```
void timexPost
(
    int    i,           /* function number in list (0..3) */
    FUNCPTR func,      /* function to be added (NULL if to be deleted) */
    int    arg1,       /* first of up to 8 args to call function with */
    int    arg2,
    int    arg3,
    int    arg4,
    int    arg5,
    int    arg6,
    int    arg7,
    int    arg8
)
```

**DESCRIPTION** This routine adds or deletes functions in the list of functions to be called immediately following the timed functions. A maximum of four functions may be included. Up to eight arguments may be passed to each function.

**RETURNS** N/A

**SEE ALSO** `timexLib`

---

## timexPre()

**NAME** `timexPre()` – specify functions to be called prior to timing

**SYNOPSIS**

```
void timexPre
(
    int      i,           /* function number in list (0..3) */
    FUNCPTR  func,       /* function to be added (NULL if to be deleted) */
    int      arg1,       /* first of up to 8 args to call function with */
    int      arg2,
    int      arg3,
    int      arg4,
    int      arg5,
    int      arg6,
    int      arg7,
    int      arg8
)
```

**DESCRIPTION** This routine adds or deletes functions in the list of functions to be called immediately prior to the timed functions. A maximum of four functions may be included. Up to eight arguments may be passed to each function.

**RETURNS** N/A

**SEE ALSO** `timexLib`

---

## timexShow()

**NAME** `timexShow()` – display the list of function calls to be timed

**SYNOPSIS**

```
void timexShow (void)
```

**DESCRIPTION** This routine displays the current list of function calls to be timed. These lists are created by calls to `timexPre()`, `timexFunc()`, and `timexPost()`.

**RETURNS** N/A

**SEE ALSO** `timexLib`, `timexPre()`, `timexFunc()`, `timexPost()`

---

## tmpfile()

<b>NAME</b>	<b>tmpfile()</b> – create a temporary binary file (Unimplemented) (ANSI)
<b>SYNOPSIS</b>	<b>FILE * tmpfile (void)</b>
<b>DESCRIPTION</b>	This routine is not be implemented because VxWorks does not close all open files at task exit.
<b>INCLUDE FILES</b>	<b>stdio.h</b>
<b>RETURNS</b>	<b>NULL</b>
<b>SEE ALSO</b>	<b>ansiStdio</b>

---

## tmpnam()

<b>NAME</b>	<b>tmpnam()</b> – generate a temporary file name (ANSI)
<b>SYNOPSIS</b>	<pre>char * tmpnam (     char * s                /* name buffer */ )</pre>
<b>DESCRIPTION</b>	<p>This routine generates a string that is a valid file name and not the same as the name of an existing file. It generates a different string each time it is called, up to <b>TMP_MAX</b> times.</p> <p>If the argument is a null pointer, <b>tmpnam()</b> leaves its result in an internal static object and returns a pointer to that object. Subsequent calls to <b>tmpnam()</b> may modify the same object. If the argument is not a null pointer, it is assumed to point to an array of at least <b>L_tmpnam</b> chars; <b>tmpnam()</b> writes its result in that array and returns the argument as its value.</p>
<b>INCLUDE FILES</b>	<b>stdio.h</b>
<b>RETURNS</b>	A pointer to the file name.
<b>SEE ALSO</b>	<b>ansiStdio</b>

## **tolower()**

<b>NAME</b>	<b>tolower()</b> – convert an upper-case letter to its lower-case equivalent (ANSI)
<b>SYNOPSIS</b>	<pre>int tolower (     int c                /* character to convert */ )</pre>
<b>DESCRIPTION</b>	This routine converts an upper-case letter to the corresponding lower-case letter.
<b>INCLUDE FILES</b>	<code>ctype.h</code>
<b>RETURNS</b>	If <i>c</i> is an upper-case letter, it returns the lower-case equivalent; otherwise, it returns the argument unchanged.
<b>SEE ALSO</b>	<code>ansiCtype</code>

---

## **toupper()**

<b>NAME</b>	<b>toupper()</b> – convert a lower-case letter to its upper-case equivalent (ANSI)
<b>SYNOPSIS</b>	<pre>int toupper (     int c                /* character to convert */ )</pre>
<b>DESCRIPTION</b>	This routine converts a lower-case letter to the corresponding upper-case letter.
<b>INCLUDE FILES</b>	<code>ctype.h</code>
<b>RETURNS</b>	If <i>c</i> is a lower-case letter, it returns the upper-case equivalent; otherwise, it returns the argument unchanged.
<b>SEE ALSO</b>	<code>ansiCtype</code>

---

## tr()

<b>NAME</b>	<code>tr()</code> – resume a task
<b>SYNOPSIS</b>	<pre>void tr (     int taskNameOrId      /* task name or task ID */ )</pre>
<b>DESCRIPTION</b>	This command resumes the execution of a suspended task. It simply calls <code>taskResume()</code> .
<b>RETURNS</b>	N/A
<b>SEE ALSO</b>	<code>usrLib</code> , <code>ts()</code> , <code>taskResume()</code> , <i>VxWorks Programmer's Guide: Target Shell</i> , <code>windsh</code> , <i>Tornado User's Guide: Shell</i>

---

## trgAdd()

<b>NAME</b>	<code>trgAdd()</code> – add a new trigger to the trigger list
<b>SYNOPSIS</b>	<pre>TRIGGER_ID trgAdd (     event_t   event,     int       status,     int       contextType,     UINT32    contextId,     OBJ_ID    objId,     int       conditional,     int       condType,     int *     condEx1,     int       condOp,     int       condEx2,     BOOL      disable,     TRIGGER * chain,     int       actionType,     FUNCPTR   actionFunc,     BOOL      actionDef,     int       actionArg )</pre>

**trgAdd()****DESCRIPTION**

This routine creates a new trigger and adds it to the proper trigger list. Parameters:

*event*

as defined in **eventP.h** for WindView, if given.

*status*

the initial status of the trigger (enabled or disabled).

*contextType*

the type of context where the event occurs.

*contextId*

the ID (if any) of the context where the event occurs.

*objectId*

if given and applicable.

*conditional*

the indicator that there is a condition on the trigger.

*condType*

the indicator that the condition is either a variable or a function.

*condEx1*

the first element in the comparison.

*condOp*

the type of operator (==, !=, <, <=, >, >=, |, &).

*condEx2*

the second element in the comparison (a constant).

*disable*

the indicator of whether the trigger must be disabled once it is hit.

*chain*

a pointer to another trigger associated to this one (if any).

*actionType*

the type of action associated with the trigger (none, func, lib).

*actionFunc*

the action associated with the trigger (the function).

*actionDef*

the indicator of whether the action can be deferred (deferred is the default).

*actionArg*

the argument passed to the function, if any.

Calling **trgAdd()** while triggering is enabled is not allowed and will return **NULL**.

**RETURNS**

**TRIGGER\_ID**, or **NULL** if either the trigger ID can not be allocated, or if called whilst triggering is enabled.

SEE ALSO `trgLib`, `trgDelete()`

---

## trgChainSet()

**NAME** `trgChainSet()` – chains two triggers

**SYNOPSIS**

```
STATUS trgChainSet
(
    TRIGGER_ID fromId,
    TRIGGER_ID toId
)
```

**DESCRIPTION** This routine chains two triggers together. When the first trigger fires, it calls `trgEnable()` for the second trigger. The second trigger must be created disabled in order to maintain the correct sequence.

**RETURNS** OK or ERROR.

SEE ALSO `trgLib`, `trgEnable()`

---

## trgDelete()

**NAME** `trgDelete()` – delete a trigger from the trigger list

**SYNOPSIS**

```
STATUS trgDelete
(
    TRIGGER_ID trgId
)
```

**DESCRIPTION** This routine deletes a trigger by removing it from the trigger list. It also checks that no other triggers are still active. If there are no active triggers and triggering is still on, it turns triggering off.

**RETURNS** OK, or ERROR if the trigger is not found.

SEE ALSO `trgLib`, `trgAdd()`

## trgDisable()

<b>NAME</b>	<b>trgDisable()</b> – turn a trigger off
<b>SYNOPSIS</b>	<pre><b>STATUS</b> <b>trgDisable</b> (     <b>TRIGGER_ID</b> <b>trgId</b> )</pre>
<b>DESCRIPTION</b>	This routine disables a trigger. It also checks to see if there are triggers still active. If this is the last active trigger it sets triggering off.
<b>RETURNS</b>	OK, or ERROR if the trigger ID is not found.
<b>SEE ALSO</b>	<b>trgLib</b> , <b>trgEnable()</b>

---

## trgEnable()

<b>NAME</b>	<b>trgEnable()</b> – enable a trigger
<b>SYNOPSIS</b>	<pre><b>STATUS</b> <b>trgEnable</b> (     <b>TRIGGER_ID</b> <b>trgId</b> )</pre>
<b>DESCRIPTION</b>	This routine enables a trigger that has been created with <b>trgAdd()</b> . A counter is incremented to keep track of the total number of enabled triggers so that <b>trgDisable()</b> knows when to set triggering off. If the maximum number of enabled triggers is reached, an error is returned.
<b>RETURNS</b>	OK, or ERROR if the trigger ID is not found or if the maximum number of triggers has already been enabled.
<b>SEE ALSO</b>	<b>trgLib</b> , <b>trgDisable()</b>

---

## trgEvent()

**NAME** `trgEvent()` – trigger a user-defined event

**SYNOPSIS**

```
void trgEvent
(
    event_t evtId          /* event */
)
```

**DESCRIPTION** This routine triggers a user event. A trigger must exist and triggering must have been started with `trgOn()` or from the triggering GUI to use this routine. The *evtId* should be in the range 40000-65535.

**RETURNS** N/A

**SEE ALSO** `trgLib`, `dbgLib`, `e()`

---

## trgLibInit()

**NAME** `trgLibInit()` – initialize the triggering library

**SYNOPSIS** `STATUS trgLibInit (void)`

**DESCRIPTION** This routine initializes the trigger class. Triggers are VxWorks objects and therefore require a class to be initialized.

**RETURNS** OK or ERROR.

**SEE ALSO** `trgLib`

## trgOff()

<b>NAME</b>	<b>trgOff()</b> – set triggering off
<b>SYNOPSIS</b>	<b>void</b> <b>trgOff</b> ( <b>void</b> )
<b>DESCRIPTION</b>	This routine turns triggering off. From this time on, when an event point is hit, no search on triggers is performed.
<b>RETURNS</b>	N/A
<b>SEE ALSO</b>	<b>trgLib</b> , <b>trgOn()</b>

---

## trgOn()

<b>NAME</b>	<b>trgOn()</b> – set triggering on
<b>SYNOPSIS</b>	<b>STATUS</b> <b>trgOn</b> ( <b>void</b> )
<b>DESCRIPTION</b>	This routine activates triggering. From this time on, any time an event point is hit, a check for the presence of possible triggers is performed. Start triggering only when needed since some overhead is introduced.  <b>NOTE:</b> If <b>trgOn()</b> is called when there are no triggers in the trigger list, it immediately sets triggering off again. If <b>trgOn()</b> is called with at least one trigger in the list, triggering begins. Triggers should not be added to the list while triggering is on since this can create instability.
<b>RETURNS</b>	OK or ERROR.
<b>SEE ALSO</b>	<b>trgLib</b> , <b>trgOff()</b>

---

## trgShow()

**NAME**            trgShow() – show trigger information

**SYNOPSIS**        **STATUS** trgShow  
                  (  
                  **TRIGGER\_ID** trgId,  
                  **int**         **level**  
                  )

**DESCRIPTION**    This routine displays trigger information. If *trgId* is passed, only the summary for that trigger is displayed. If no parameter is passed, the list of existing triggers is displayed with a summary of their state. For example:

trgID	Status	EvtID	ActType	Action	Dis	Chain
0xffedfc	disabled	101	3	0x14e7a4	Y	0xffe088
0xffe088	enabled	55	1	0x10db58	Y	0x0

If *level* is 1, then more detailed information is displayed.

**EXAMPLE**        -> `trgShow trgId, 1`

**RETURNS**        OK.

**SEE ALSO**        trgShow, trgLib

---

## trgShowInit()

**NAME**            trgShowInit() – initialize the trigger show facility

**SYNOPSIS**        **void** trgShowInit (**void**)

**DESCRIPTION**    This routine links the trigger show facility into the VxWorks system. These routines are included automatically when `INCLUDE_TRIGGER_SHOW` is defined.

**RETURNS**        N/A

**SEE ALSO**        trgShow

## **trgWorkQReset()**

<b>NAME</b>	<b>trgWorkQReset()</b> – reset the trigger work queue task and queue
<b>SYNOPSIS</b>	<b>STATUS</b> <b>trgWorkQReset (void)</b>
<b>DESCRIPTION</b>	<p>When a trigger fires, if the associated action requires a function to be called in “safe” mode, a pointer to the required function will be placed on a queue known as the “triggering work queue”. A system task “<b>tActDef</b>” is spawned to action these requests at task level. Should the user have need to reset this work queue (<i>e.g.</i>, if a called task causes an exception which causes the <b>trgActDef</b> task to be <b>SUSPENDED</b>, or if the queue gets out of sync and becomes unresponsive), <b>trgWorkQReset()</b> may be called.</p> <p>Its effect is to delete the trigger work queue task and its associated resources and then recreate them. Any entries pending on the triggering work queue will be lost. Calling this function with triggering on will result in triggering being turned off before the queue reset takes place. It is the responsibility of the user to turn triggering back on.</p>
<b>RETURNS</b>	<b>OK</b> , or <b>ERROR</b> if the triggering task and its associated resources cannot be deleted and recreated.
<b>SEE ALSO</b>	<b>trgLib</b>

---

## **trunc()**

<b>NAME</b>	<b>trunc()</b> – truncate to integer
<b>SYNOPSIS</b>	<pre>double trunc (     double x /* value to truncate */ )</pre>
<b>DESCRIPTION</b>	This routine discards the fractional part of a double-precision value <i>x</i> .
<b>INCLUDE FILES</b>	<b>math.h</b>
<b>RETURNS</b>	The integer portion of <i>x</i> , represented in double-precision.
<b>SEE ALSO</b>	<b>mathALib</b>

---

## truncf()

<b>NAME</b>	<b>truncf()</b> – truncate to integer
<b>SYNOPSIS</b>	<pre>float truncf (     float x    /* value to truncate */ )</pre>
<b>DESCRIPTION</b>	This routine discards the fractional part of a single-precision value <i>x</i> .
<b>INCLUDE FILES</b>	<b>math.h</b>
<b>RETURNS</b>	The integer portion of <i>x</i> , represented in single precision.
<b>SEE ALSO</b>	<b>mathALib</b>

---

## ts()

<b>NAME</b>	<b>ts()</b> – suspend a task
<b>SYNOPSIS</b>	<pre>void ts (     int taskNameOrId    /* task name or task ID */ )</pre>
<b>DESCRIPTION</b>	This command suspends the execution of a specified task. It simply calls <b>taskSuspend()</b> .
<b>RETURNS</b>	N/A
<b>SEE ALSO</b>	<b>usrLib</b> , <b>tr()</b> , <b>taskSuspend()</b> , <i>VxWorks Programmer's Guide: Target Shell</i> , <b>windsh</b> , <i>Tornado User's Guide: Shell</i>

## tsfsUploadPathClose()

**NAME** `tsfsUploadPathClose()` – close the TSFS-socket upload path (Windview)

**SYNOPSIS**

```
void tsfsUploadPathClose
(
    UPLOAD_ID upId          /* generic upload-path descriptor */
)
```

**DESCRIPTION** This routine closes the TSFS-socket connection to the event receiver on the host.

**RETURNS** N/A

**SEE ALSO** `wvTsfsUploadPathLib`, `tsfsUploadPathCreate()`

---

## tsfsUploadPathCreate()

**NAME** `tsfsUploadPathCreate()` – open an upload path to the host using a TSFS socket (Windview)

**SYNOPSIS**

```
UPLOAD_ID tsfsUploadPathCreate
(
    char * ipAddress,      /* server's IP address in .-notation */
    short port             /* port number to bind to */
)
```

**DESCRIPTION** This routine opens a TSFS socket to the host to be used for uploading event data. After successfully establishing this connection, an `UPLOAD_ID` is returned which points to the `TSFS_UPLOAD_DESC` that is passed to `open()`, `close()`, `read()`, etc. for future operations.

**RETURNS** The `UPLOAD_ID`, or `NULL` if the connection cannot be completed or not enough memory is available.

**SEE ALSO** `wvTsfsUploadPathLib`, `tsfsUploadPathClose()`

---

## tsfsUploadPathLibInit()

**NAME** `tsfsUploadPathLibInit()` – initialize `wvTsfsUploadPathLib` library (Windview)

**SYNOPSIS** `STATUS tsfsUploadPathLibInit (void)`

**DESCRIPTION** This routine initializes `wvTsfsUploadPathLib` by pulling in the routines in this file for use with WindView. It is called during system configuration from `usrWindview.c`.

**RETURNS** OK.

**SEE ALSO** `wvTsfsUploadPathLib`

---

## tsfsUploadPathWrite()

**NAME** `tsfsUploadPathWrite()` – write to the TSFS upload path (Windview)

**SYNOPSIS**

```
int tsfsUploadPathWrite
(
    UPLOAD_ID upId,          /* generic upload-path descriptor */
    char *    pStart,        /* address of data to write */
    size_t    size           /* number of bytes of data at pStart */
)
```

**DESCRIPTION** This routine writes *size* bytes of data beginning at *pStart* to the upload path connecting the target with the host receiver.

**RETURNS** The number of bytes written, or `ERROR`.

**SEE ALSO** `wvTsfsUploadPathLib`, `tsfsUploadPathCreate()`

**tt()****tt()**

**NAME** `tt()` – display a stack trace of a task

**SYNOPSIS**

```
STATUS tt
(
    int taskNameOrId          /* task name or task ID */
)
```

**DESCRIPTION** This routine displays a list of the nested routine calls that the specified task is in. Each routine call and its parameters are shown.

If *taskNameOrId* is not specified or zero, the last task referenced is assumed. The `tt()` routine can only trace the stack of a task other than itself. For instance, when `tt()` is called from the shell, it cannot trace the shell's stack.

**EXAMPLE**

```
-> tt "logTask"
3ab92 _vxTaskEntry   +10 : _logTask (0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0)
ee6e _logTask       +12 : _read (5, 3f8a10, 20)
d460 _read          +10 : _iosRead (5, 3f8a10, 20)
e234 _iosRead       +9c : _pipeRead (3fce1c, 3f8a10, 20)
23978 _pipeRead     +24 : _semTake (3f8b78)
value = 0 = 0x0
```

This indicates that `logTask()` is currently in `semTake()` (with one parameter) and was called by `pipeRead()` (with three parameters), which was called by `iosRead()` (with three parameters), and so on.

---

**WARNING:** In order to do the trace, some assumptions are made. In general, the trace will work for all C language routines and for assembly language routines that start with a LINK instruction. Some C compilers require specific flags to generate the LINK first. Most VxWorks assembly language routines include LINK instructions for this reason. The trace facility may produce inaccurate results or fail completely if the routine is written in a language other than C, the routine's entry point is non-standard, or the task's stack is corrupted. Also, all parameters are assumed to be 32-bit quantities, so structures passed as parameters will be displayed as *long* integers.

---

**RETURNS** OK, or ERROR if the task does not exist.

**SEE ALSO** `dbgLib`, *VxWorks Programmer's Guide: Target Shell*, `windsh`, *Tornado User's Guide: Shell*

---

## ttyDevCreate()

**NAME** `ttyDevCreate()` – create a VxWorks device for a serial channel

**SYNOPSIS**

```
STATUS ttyDevCreate
(
    char *      name,          /* name to use for this device */
    SIO_CHAN * pSioChan,     /* pointer to core driver structure */
    int        rdBufSize,    /* read buffer size, in bytes */
    int        wrtBufSize    /* write buffer size, in bytes */
)
```

**DESCRIPTION** This routine creates a device on a specified serial channel. Each channel to be used should have exactly one device associated with it by calling this routine.

For instance, to create the device `/tyCo/0`, with buffer sizes of 512 bytes, the proper call would be:

```
    ttyDevCreate ("/tyCo/0", pSioChan, 512, 512);
```

Where `pSioChan` is the address of the underlying `SIO_CHAN` serial channel descriptor (defined in `sioLib.h`). This routine is typically called by `usrRoot()` in `usrConfig.c`

**RETURNS** OK, or `ERROR` if the driver is not installed, or the device already exists.

**SEE ALSO** `ttyDrv`

---

## ttyDrv()

**NAME** `ttyDrv()` – initialize the tty driver

**SYNOPSIS** `STATUS` `ttyDrv (void)`

**DESCRIPTION** This routine initializes the tty driver, which is the OS interface to core serial channel(s). Normally, it is called by `usrRoot()` in `usrConfig.c`.

After this routine is called, `ttyDevCreate()` is typically called to bind serial channels to VxWorks devices.

**RETURNS** OK, or `ERROR` if the driver cannot be installed.

**SEE ALSO** `ttyDrv`

---

## tyAbortFuncSet()

<b>NAME</b>	<b>tyAbortFuncSet()</b> – set the abort function
<b>SYNOPSIS</b>	<pre>void tyAbortFuncSet (     FUNCPTR func           /* routine to call when abort char received */ )</pre>
<b>DESCRIPTION</b>	<p>This routine sets the function that will be called when the abort character is received on a tty. There is only one global abort function, used for any tty on which <b>OPT_ABORT</b> is enabled. When the abort character is received from a tty with <b>OPT_ABORT</b> set, the function specified in <i>func</i> will be called, with no parameters, from interrupt level.</p> <p>Setting an abort function of <b>NULL</b> will disable the abort function.</p>
<b>RETURNS</b>	N/A
<b>SEE ALSO</b>	<b>tyLib</b> , <b>tyAbortSet()</b>

---

## tyAbortSet()

<b>NAME</b>	<b>tyAbortSet()</b> – change the abort character
<b>SYNOPSIS</b>	<pre>void tyAbortSet (     char ch                /* char to be abort */ )</pre>
<b>DESCRIPTION</b>	<p>This routine sets the abort character to <i>ch</i>. The default abort character is CTRL-C.</p> <p>Typing the abort character to any device whose <b>OPT_ABORT</b> option is set will cause the shell task to be killed and restarted. Note that the character set by this routine applies to all devices whose handlers use the standard tty package <b>tyLib</b>.</p>
<b>RETURNS</b>	N/A
<b>SEE ALSO</b>	<b>tyLib</b> , <b>tyAbortFuncSet()</b>

---

## tyBackspaceSet()

**NAME** `tyBackspaceSet()` – change the backspace character

**SYNOPSIS**

```
void tyBackspaceSet
(
    char ch                /* char to be backspace */
)
```

**DESCRIPTION** This routine sets the backspace character to *ch*. The default backspace character is CTRL-H.

Typing the backspace character to any device operating in line protocol mode (`OPT_LINE` set) will cause the previous character typed to be deleted, up to the beginning of the current line. Note that the character set by this routine applies to all devices whose handlers use the standard tty package `tyLib`.

**RETURNS** N/A

**SEE ALSO** `tyLib`

---

## tyDeleteLineSet()

**NAME** `tyDeleteLineSet()` – change the line-delete character

**SYNOPSIS**

```
void tyDeleteLineSet
(
    char ch                /* char to be line-delete */
)
```

**DESCRIPTION** This routine sets the line-delete character to *ch*. The default line-delete character is CTRL-U.

Typing the delete character to any device operating in line protocol mode (`OPT_LINE` set) will cause all characters in the current line to be deleted. Note that the character set by this routine applies to all devices whose handlers use the standard tty package `tyLib`.

**RETURNS** N/A

**SEE ALSO** `tyLib`

---

## tyDevInit()

**NAME** tyDevInit() – initialize the tty device descriptor

**SYNOPSIS**

```
STATUS tyDevInit
(
    TY_DEV_ID pTyDev,          /* ptr to tty dev descriptor to init */
    int      rdBufSize,       /* size of read buffer in bytes */
    int      wrtBufSize,      /* size of write buffer in bytes */
    FUNCPTR  txStartup        /* device transmit start-up routine */
)
```

**DESCRIPTION** This routine initializes a tty device descriptor according to the specified parameters. The initialization includes allocating read and write buffers of the specified sizes from the memory pool, and initializing their respective buffer descriptors. The semaphores are initialized and the write semaphore is given to enable writers. Also, the transmitter start-up routine pointer is set to the specified routine. All other fields in the descriptor are zeroed.

This routine should be called only by serial drivers.

**RETURNS** OK, or ERROR if there is not enough memory to allocate data structures.

**SEE ALSO** tyLib

---

## tyDevRemove()

**NAME** tyDevRemove() – remove the tty device descriptor

**SYNOPSIS**

```
STATUS tyDevRemove
(
    TY_DEV_ID pTyDev          /* ptr to tty dev descriptor to remove */
)
```

**DESCRIPTION** This routine removes an existing tty device descriptor. It releases the read and write buffers and the descriptor data structure.

**RETURNS** OK, or ERROR if expected data structures are not found

**SEE ALSO** tyLib

---

## tyEOFSet()

**NAME** `tyEOFSet()` – change the end-of-file character

**SYNOPSIS**

```
void tyEOFSet
(
    char ch                /* char to be EOF */
)
```

**DESCRIPTION** This routine sets the EOF character to *ch*. The default EOF character is CTRL-D. Typing the EOF character to any device operating in line protocol mode (`OPT_LINE` set) will cause no character to be entered in the current line, but will cause the current line to be terminated (thus without a newline character). The line is made available to reading tasks. Thus, if the EOF character is the first character input on a line, a line length of zero characters is returned to the reader. This is the standard end-of-file indication on a read call. Note that the EOF character set by this routine will apply to all devices whose handlers use the standard `ty` package `tyLib`.

**RETURNS** N/A

**SEE ALSO** `tyLib`

---

## tyIoctl()

**NAME** `tyIoctl()` – handle device control requests

**SYNOPSIS**

```
STATUS tyIoctl
(
    TY_DEV_ID pTyDev,      /* ptr to device to control */
    int request,          /* request code */
    int arg                /* some argument */
)
```

**DESCRIPTION** This routine handles `ioctl()` requests for tty devices. The I/O control functions for tty devices are described in the manual entry for `tyLib`.

**BUGS** In line protocol mode (`OPT_LINE` option set), the `FIONREAD` function actually returns the number of characters available plus the number of lines in the buffer. Thus, if five lines consisting of just `NEWLINEs` were in the input buffer, the `FIONREAD` function would return the value ten (five characters + five lines).

**tyIRd()**

**RETURNS** OK or ERROR.

**SEE ALSO** tyLib

---

## tyIRd()

**NAME** tyIRd() – interrupt-level input

**SYNOPSIS**

```

STATUS tyIRd
(
    TY_DEV_ID pTyDev,          /* ptr to tty device descriptor */
    char      inchar          /* character read */
)

```

**DESCRIPTION** This routine handles interrupt-level character input for tty devices. A device driver calls this routine when it has received a character. This routine adds the character to the ring buffer for the specified device, and gives a semaphore if a task is waiting for it.

This routine also handles all the special characters, as specified in the option word for the device, such as X-on, X-off, NEWLINE, or backspace.

**RETURNS** OK, or ERROR if the ring buffer is full.

**SEE ALSO** tyLib

---

## tyITx()

**NAME** tyITx() – interrupt-level output

**SYNOPSIS**

```

STATUS tyITx
(
    TY_DEV_ID pTyDev,          /* pointer to tty device descriptor */
    char *    pChar           /* where to put character to be output */
)

```

**DESCRIPTION** This routine gets a single character to be output to a device. It looks at the ring buffer for *pTyDev* and gives the caller the next available character, if there is one. The character to be output is copied to *pChar*.

**RETURNS** OK if there are more characters to send, or **ERROR** if there are no more characters.

**SEE ALSO** tyLib

---

## tyMonitorTrapSet()

**NAME** tyMonitorTrapSet() – change the trap-to-monitor character

**SYNOPSIS**

```
void tyMonitorTrapSet
(
    char ch                /* char to be monitor trap */
)
```

**DESCRIPTION** This routine sets the trap-to-monitor character to *ch*. The default trap-to-monitor character is CTRL-X.

Typing the trap-to-monitor character to any device whose **OPT\_MON\_TRAP** option is set will cause the resident ROM monitor to be entered, if one is present. Once the ROM monitor is entered, the normal multitasking system is halted.

Note that the trap-to-monitor character set by this routine will apply to all devices whose handlers use the standard tty package **tyLib**. Also note that not all systems have a monitor trap available.

**RETURNS** N/A

**SEE ALSO** tyLib

---

## tyRead()

**NAME** tyRead() – do a task-level read for a tty device

**SYNOPSIS**

```
int tyRead
(
    TY_DEV_ID pTyDev,      /* device to read */
    char *    buffer,      /* buffer to read into */
    int      maxbytes      /* maximum length of read */
)
```

**tyWrite()**

- DESCRIPTION** This routine handles the task-level portion of the tty handler's read function. It reads into the buffer up to *maxbytes* available bytes.
- This routine should only be called from serial device drivers.
- RETURNS** The number of bytes actually read into the buffer.
- SEE ALSO** **tyLib**

---

## tyWrite()

**NAME** **tyWrite()** – do a task-level write for a tty device

**SYNOPSIS**

```
int tyWrite
(
    TY_DEV_ID pTyDev,      /* ptr to device structure */
    char *    buffer,      /* buffer of data to write */
    int      nbytes        /* number of bytes in buffer */
)
```

- DESCRIPTION** This routine handles the task-level portion of the tty handler's write function.
- RETURNS** The number of bytes actually written to the device.
- SEE ALSO** **tyLib**

---

## udpShowInit()

<b>NAME</b>	<code>udpShowInit()</code> – initialize UDP show routines
<b>SYNOPSIS</b>	<code>void udpShowInit (void)</code>
<b>DESCRIPTION</b>	This routine links the UDP show facility into the VxWorks system. These routines are included automatically if <code>INCLUDE_NET_SHOW</code> and <code>INCLUDE_UDP</code> are defined.
<b>RETURNS</b>	N/A
<b>SEE ALSO</b>	<code>udpShow</code>

---

## udpstatShow()

<b>NAME</b>	<code>udpstatShow()</code> – display statistics for the UDP protocol
<b>SYNOPSIS</b>	<code>void udpstatShow (void)</code>
<b>DESCRIPTION</b>	This routine displays statistics for the UDP protocol.
<b>RETURNS</b>	N/A
<b>SEE ALSO</b>	<code>udpShow</code>

---

## ungetc()

<b>NAME</b>	<code>ungetc()</code> – push a character back into an input stream (ANSI)
<b>SYNOPSIS</b>	<pre>int ungetc (     int    c,                /* character to push */     FILE * fp                /* input stream */ )</pre>
<b>DESCRIPTION</b>	This routine pushes a character <code>c</code> (converted to an <b>unsigned char</b> ) back into the specified input stream. The pushed-back characters will be returned by subsequent reads on that

stream in the reverse order of their pushing. A successful intervening call on the stream to a file positioning function (**fseek()**, **fsetpos()**, or **rewind()**) discards any pushed-back characters for the stream. The external storage corresponding to the stream is unchanged.

One character of push-back is guaranteed. If **ungetc()** is called too many times on the same stream without an intervening read or file positioning operation, the operation may fail.

If the value of *c* equals EOF, the operation fails and the input stream is unchanged.

A successful call to **ungetc()** clears the end-of-file indicator for the stream. The value of the file position indicator for the stream after reading or discarding all pushed-back characters is the same as it was before the character were pushed back. For a text stream, the value of its file position indicator after a successful call to **ungetc()** is unspecified until all pushed-back characters are read or discarded. For a binary stream, the file position indicator is decremented by each successful call to **ungetc()**; if its value was zero before a call, it is indeterminate after the call.

**INCLUDE**            **stdio.h**

**RETURNS**            The pushed-back character after conversion, or EOF if the operation fails.

**SEE ALSO**            **ansiStdio, getc(), fgetc()**

---

## unixDiskDevCreate()

**NAME**                **unixDiskDevCreate()** – create a UNIX disk device

**SYNOPSIS**            **BLK\_DEV \*unixDiskDevCreate**  
                  (  
                  **char \* unixFile,**            **/\* name of the UNIX file \*/**  
                  **int    bytesPerBlk,**            **/\* number of bytes per block \*/**  
                  **int    blksPerTrack,**        **/\* number of blocks per track \*/**  
                  **int    nBlocks**            **/\* number of blocks on this device \*/**  
                  )

**DESCRIPTION**        This routine creates a UNIX disk device.

The *unixFile* parameter specifies the name of the UNIX file to use for the disk device.

The *bytesPerBlk* parameter specifies the size of each logical block on the disk. If *bytesPerBlk* is zero, 512 is the default.

The *blksPerTrack* parameter specifies the number of blocks on each logical track of the disk. If *blksPerTrack* is zero, the count of blocks per track is set to *nBlocks* (i.e., the disk is defined as having only one track).

The *nBlocks* parameter specifies the size of the disk, in blocks. If *nBlocks* is zero, a default size is used. The default is calculated as the size of the UNIX disk divided by the number of bytes per block.

This routine is only applicable to VxSim for Solaris and VxSim for HP.

**RETURNS** A pointer to block device (BLK\_DEV) structure, or NULL, if unable to open the UNIX disk.

**SEE ALSO** [unixDrv](#)

---

## unixDiskInit()

**NAME** [unixDiskInit\(\)](#) – initialize a dosFs disk on top of UNIX

**SYNOPSIS**

```
void unixDiskInit
(
    char * unixFile,          /* UNIX file name */
    char * volName,          /* dosFs name */
    int  diskSize            /* number of bytes */
)
```

**DESCRIPTION** This routine provides some convenience for a user wanting to create a UNIX disk-based dosFs file system under VxWorks. The user only specifies the UNIX file to use, the dosFs volume name, and the size of the volume in bytes, if the UNIX file needs to be created.

This routine is only applicable to VxSim for Solaris and VxSim for HP.

**RETURNS** N/A

**SEE ALSO** [unixDrv](#)

## unixDrv()

<b>NAME</b>	<b>unixDrv()</b> – install UNIX disk driver
<b>SYNOPSIS</b>	<b>STATUS</b> <b>unixDrv</b> ( <b>void</b> )
<b>DESCRIPTION</b>	Used in <b>usrConfig.c</b> to cause the UNIX disk driver to be linked in when building VxWorks. Otherwise, it is not necessary to call this routine before using the UNIX disk driver.  This routine is only applicable to VxSim for Solaris and VxSim for HP.
<b>RETURNS</b>	OK (always).
<b>SEE ALSO</b>	<b>unixDrv</b>

---

## unld()

<b>NAME</b>	<b>unld()</b> – unload an object module by specifying a file name or module ID
<b>SYNOPSIS</b>	<b>STATUS</b> <b>unld</b> ( <b>void</b> * <b>nameOrId</b> ,                    /* name or ID of the object module file */ <b>int</b> <b>options</b> )
<b>DESCRIPTION</b>	This routine unloads the specified object module from the system. The module can be specified by name or by module ID. For a.out and ECOFF format modules, unloading does the following: <ol style="list-style-type: none"><li>(1) It frees the space allocated for text, data, and BSS segments, unless <b>loadModuleAt()</b> was called with specific addresses, in which case the user is responsible for freeing the space.</li><li>(2) It removes all symbols associated with the object module from the system symbol table.</li><li>(3) It removes the module descriptor from the module list.</li></ol> For other modules of other formats, unloading has similar effects.  Before any modules are unloaded, all breakpoints in the system are deleted. If you need to keep breakpoints, set the options parameter to <b>UNLD_KEEP_BREAKPOINTS</b> . No breakpoints can be set in code that is unloaded.

This routine is a **shell command**. That is, it is designed to be used only in the shell, and not in code running on the target. In future releases, calling **unld()** directly from code may not be supported.

**RETURNS** OK or ERROR.

**SEE ALSO** **unldLib**, *VxWorks Programmer's Guide: Target Shell*, **windsh**, *Tornado User's Guide: Shell*

---

## unldByGroup()

**NAME** **unldByGroup()** – unload an object module by specifying a group number

**SYNOPSIS**

```
STATUS unldByGroup
(
    UINT16 group,          /* group number to unload */
    int options           /* options, currently unused */
)
```

**DESCRIPTION** This routine unloads an object module that has a group number matching *group*. See the manual entries for **unld()** or **unldLib** for more information on module unloading.

**RETURNS** OK or ERROR.

**SEE ALSO** **unldLib**, **unld()**

---

## unldByModuleId()

**NAME** **unldByModuleId()** – unload an object module by specifying a module ID

**SYNOPSIS**

```
STATUS unldByModuleId
(
    MODULE_ID moduleId,   /* module ID to unload */
    int options
)
```

**DESCRIPTION** This routine unloads an object module that has a module ID matching *moduleId*. See the manual entries for **unld()** or **unldLib** for more information on module unloading.

**RETURNS** OK or ERROR.

**SEE ALSO** **unldLib**, **unld()**



## unldByNameAndPath()

**NAME** `unldByNameAndPath()` – unload an object module by specifying a name and path

**SYNOPSIS**

```
STATUS unldByNameAndPath
(
    char * name,           /* name of the object module to unload */
    char * path,          /* path to the object module to unload */
    int  options          /* options, currently unused */
)
```

**DESCRIPTION** This routine unloads an object module specified by *name* and *path*.  
See the manual entries for `unld()` or `unldLib` for more information on module unloading.

**RETURNS** OK or ERROR.

**SEE ALSO** `unldLib`, `unld()`

---

## unlink()

**NAME** `unlink()` – delete a file (POSIX)

**SYNOPSIS**

```
STATUS unlink
(
    char * name           /* name of the file to remove */
)
```

**DESCRIPTION** This routine deletes a specified file. It performs the same function as `remove()` and is provided for POSIX compatibility.

**RETURNS** OK if there is no delete routine for the device or the driver returns OK; ERROR if there is no such device or the driver returns ERROR.

**SEE ALSO** `ioLib`, `remove()`

---

## usrAtaConfig()

**NAME** `usrAtaConfig()` – mount a DOS file system from an ATA hard disk or a CDROM

**SYNOPSIS**

```
STATUS usrAtaConfig
(
    int    ctrl,           /* 0: primary address, 1: secondary address */
    int    drive,         /* drive number of hard disk (0 or 1) */
    char * devNames       /* mount points for each partition */
)
```

**DESCRIPTION** file system from an ATAPI CDROM drive

This routine mounts a DOS file system from an ATA hard disk. Parameters:

*drive*

the drive number of the hard disk; 0 is C: and 1 is D:.

*devName*

the mount point for all partitions which are expected to be present on the disk, separated with commas, for example `/ata0,/ata1` or `C:,D:`. Blanks are not allowed in this string. If the drive is an ATAPI CDROM drive, then the CDROM file system is specified by appending `(cdrom)` after the mount point name. For example, a CDROM drive could be specified as `/cd(cdrom)`.

---

**NOTE:** Because VxWorks does not support creation of partition tables, hard disks formatted and initialized on VxWorks are not compatible with DOS machines. This routine does not refuse to mount a hard disk that was initialized on VxWorks. Up to 8 disk partitions are supported.

---

**RETURNS** OK or ERROR.

**SEE ALSO** `usrAta`, `src/config/usrAta.c`, *VxWorks Programmer's Guide: I/O System, Local File Systems, Intel i386/i486/Pentium*

---

## usrAtaInit()

**NAME** `usrAtaInit()` – initialize the hard disk driver

**SYNOPSIS** `void usrAtaInit (void)`

**DESCRIPTION** This routine is called from `usrConfig.c` to initialize the hard drive.

**SEE ALSO** `usrAta`

---

## usrClock()

**NAME** `usrClock()` – user-defined system clock interrupt routine

**SYNOPSIS** `void usrClock ()`

**DESCRIPTION** This routine is called at interrupt level on each clock interrupt. It is installed by `usrRoot()` with a `sysClkConnect()` call. It calls all the other packages that need to know about clock ticks, including the kernel itself.

If the application needs anything to happen at the system clock interrupt level, it can be added to this routine.

**RETURNS** N/A

**SEE ALSO** `usrConfig`

---

## usrFdConfig()

**NAME** `usrFdConfig()` – mount a DOS file system from a floppy disk

**SYNOPSIS** `STATUS usrFdConfig`

```
(
  int   drive,           /* drive number of floppy disk (0 - 3) */
  int   type,            /* type of floppy disk */
  char * fileName       /* mount point */
)
```

- DESCRIPTION** This routine mounts a DOS file system from a floppy disk device. The *drive* parameter is the drive number of the floppy disk; valid values are 0 to 3. The *type* parameter specifies the type of diskette, which is described in the structure table **fdTypes[]** in **sysLib.c**. *type* is an index to the table. Currently the table contains two diskette types:
- A *type* of 0 indicates the first entry in the table (3.5" 2HD, 1.44MB);
  - A *type* of 1 indicates the second entry in the table (5.25" 2HD, 1.2MB).
- The *fileName* parameter is the mount point, *e.g.*, **/fd0/**.
- RETURNS** OK or ERROR.
- SEE ALSO** **usrFd**, *VxWorks Programmer's Guide: I/O System, Local File Systems, Intel i386/i486 Appendix*

---

## usrFdiskPartCreate()

**NAME** **usrFdiskPartCreate()** – create an FDISK-like partition table on a disk

**SYNOPSIS**

```

STATUS usrFdiskPartCreate
(
    CBIO_DEV_ID cDev,      /* device representing the entire disk */
    int         nPart,     /* how many partitions needed, default=1, max=4 */
    int         size1,     /* space percentage for second partition */
    int         size2,     /* space percentage for third partition */
    int         size3      /* space percentage for fourth partition */
)

```

**DESCRIPTION** This function may be used to create a basic PC partition table. Such partition table however is not intended to be compatible with other operating systems, it is intended for disks connected to a VxWorks target, but without the access to a PC which may be used to create the partition table.

This function is capable of creating only one partition table - the MBR, and will not create any Bootable or Extended partitions. Therefore, 4 partitions are supported.

*dev* is a CBIO device handle for an entire disk, *e.g.*, a handle returned by **dcacheDevCreate()**, or if **dpartCbio** is used, it can be either the Master partition manager handle, or the one of the 0th partition if the disk does not contain a partition table at all.

The *nPart* argument contains the number of partitions to create. If *nPart* is 0 or 1, then a single partition covering the entire disk is created. If *nPart* is between 2 and 4, then the arguments *size1*, *size2* and *size3* contain the *percentage* of disk space to be assigned to the



2nd, 3rd, and 4th partitions respectively. The first partition (partition 0) will be assigned the remainder of space left (space hog).

Partition sizes will be round down to be multiple of whole tracks so that partition Cylinder/Head/Track fields will be initialized as well as the LBA fields. Although the CHS fields are written they are not used in VxWorks, and can not be guaranteed to work correctly on other systems.

**RETURNS** OK or ERROR writing a partition table to disk

**SEE ALSO** `usrFdiskPartLib`

---

## usrFdiskPartRead()

**NAME** `usrFdiskPartRead()` – read an FDISK-style partition table

**SYNOPSIS**

```
STATUS usrFdiskPartRead
(
    CBIO_DEV_ID      cDev,      /* device from which to read blocks */
    PART_TABLE_ENTRY * pPartTab, /* table where to fill results */
    int              nPart      /* # of entries in pPartTable */
)
```

**DESCRIPTION** This function will read and decode a PC formatted partition table on a disk, and fill the appropriate partition table array with the resulting geometry, which should be used by the `dpartCbio` partition manager to access a partitioned disk with a shared disk cache.

**EXAMPLE** The following example shows how a hard disk which is expected to have up to two partitions might be configured, assuming the physical level initialization resulted in the `blkIoDevId` handle:

```
devCbio = dcacheDevCreate( blkIoDevId, 0, 0x20000, "Hard Disk");
mainDevId = dpartDevCreate( devCbio, 2, usrFdiskPartRead )
dosFsDevCreate( "/disk0a", dpartPartGet (mainDevId, 0), 0,0,0);
dosFsDevCreate( "/disk0b", dpartPartGet (mainDevId, 1), 0,0,0);
```

**RETURNS** OK or ERROR if partition table is corrupt

**SEE ALSO** `usrFdiskPartLib`

---

## usrFdiskPartShow()

**NAME** `usrFdiskPartShow()` – parse and display partition data

**SYNOPSIS**

```
STATUS usrFdiskPartShow
(
    CBIO_DEV_ID cbio,          /* device CBIO handle */
    block_t     extPartOffset, /* user should pass zero */
    block_t     currentOffset, /* user should pass zero */
    int         extPartLevel  /* user should pass zero */
)
```

**DESCRIPTION** This routine is intended to be user callable.

A device dependent partition table show routine. This routine outputs formatted data for all partition table fields for every partition table found on a given disk, starting with the MBR sectors partition table. This code can be removed to reduce code size by undefining: `INCLUDE_PART_SHOW` and rebuilding this library and linking to the new library.

This routine takes three arguments. First, a CBIO pointer (assigned for the entire physical disk) usually obtained from `dcacheDevCreate()`. It also takes two `block_t` type arguments and one signed int, the user shall pass zero in these parameters.

For example:

```
sp usrFdiskPartShow (pCbio,0,0,0)
```

Developers may use *sizearch* to view code size.

**RETURNS** OK or ERROR

**SEE ALSO** `usrFdiskPartLib`

---

## usrIdeConfig()

<b>NAME</b>	<b>usrIdeConfig()</b> – mount a DOS file system from an IDE hard disk
<b>SYNOPSIS</b>	<pre><b>STATUS</b> usrIdeConfig (     int    drive,           /* drive number of hard disk (0 or 1) */     char * fileName       /* mount point */ )</pre>
<b>DESCRIPTION</b>	<p>This routine mounts a DOS file system from an IDE hard disk.</p> <p>The <i>drive</i> parameter is the drive number of the hard disk; 0 is <b>C:</b> and 1 is <b>D:</b>.</p> <p>The <i>fileName</i> parameter is the mount point, <i>e.g.</i>, <b>/ide0/</b>.</p> <hr/> <p><b>NOTE:</b> Because VxWorks does not support partitioning, hard disks formatted and initialized on VxWorks are not compatible with DOS machines. This routine does not refuse to mount a hard disk that was initialized on VxWorks. The hard disk is assumed to have only one partition with a partition record in sector 0.</p> <hr/>
<b>RETURNS</b>	OK or ERROR.
<b>SEE ALSO</b>	<b>usrIde</b> , <i>VxWorks Programmer's Guide: I/O System, Local File Systems, Intel i386/i486 Appendix</i>

---

## usrInit()

<b>NAME</b>	<b>usrInit()</b> – user-defined system initialization routine
<b>SYNOPSIS</b>	<pre><b>void</b> usrInit (     int startType )</pre>
<b>DESCRIPTION</b>	<p>This is the first C code executed after the system boots. This routine is called by the assembly language start-up routine <b>sysInit()</b> which is in the <b>sysALib</b> module of the target-specific directory. It is called with interrupts locked out. The kernel is not multitasking at this point.</p> <p>This routine starts by clearing BSS; thus all variables are initialized to 0, as per the C specification. It then initializes the hardware by calling <b>sysHwInit()</b>, sets up the</p>

interrupt/exception vectors, and starts kernel multitasking with **usrRoot()** as the root task.

**RETURNS** N/A

**SEE ALSO** **usrConfig**, **kernelLib**

---

## usrRoot()

**NAME** **usrRoot()** – the root task

**SYNOPSIS**

```
void usrRoot
(
    char *   pMemPoolStart, /* start of system memory partition */
    unsigned memPoolSize   /* initial size of mem pool */
)
```

**DESCRIPTION** This is the first task to run under the multitasking kernel. It performs all final initialization and then starts other tasks.

It initializes the I/O system, installs drivers, creates devices, and sets up the network, etc., as necessary for a particular configuration. It may also create and load the system symbol table, if one is to be included. It may then load and spawn additional tasks as needed. In the default configuration, it simply initializes the VxWorks shell.

**RETURNS** N/A

**SEE ALSO** **usrConfig**

---

## usrScsiConfig()

**NAME** **usrScsiConfig()** – configure SCSI peripherals

**SYNOPSIS** **STATUS** **usrScsiConfig (void)**

**DESCRIPTION** This code configures the SCSI disks and other peripherals on a SCSI controller chain.

The macro **SCSI\_AUTO\_CONFIG** will include code to scan all possible device/lun id's and to configure a **scsiPhysDev** structure for each device found. Of course this doesn't include final configuration for disk partitions, floppy configuration parameters, or tape system

**uswab()**

setup. All of these actions must be performed by user code, either through **sysScsiConfig()**, the startup script, or by the application program.

The user may customize this code on a per BSP basis using the **SYS\_SCSI\_CONFIG** macro. If defined, then this routine will call the routine **sysScsiConfig()**. That routine is to be provided by the BSP, either in **sysLib.c** or **sysScsi.c**. If **SYS\_SCSI\_CONFIG** is not defined, then **sysScsiConfig()** will not be called as part of this routine.

An example **sysScsiConfig()** routine can be found in **target/src/config/usrScsi.c**. The example code contains sample configurations for a hard disk, a floppy disk and a tape unit.

**RETURNS** OK or ERROR.

**SEE ALSO** **usrScsi**, *VxWorks Programmer's Guide: I/O System, Local File Systems*

---

## uswab()

**NAME** **uswab()** – swap bytes with buffers that are not necessarily aligned

**SYNOPSIS**

```
void uswab
(
    char * source,           /* pointer to source buffer */
    char * destination,     /* pointer to destination buffer */
    int  nbytes             /* number of bytes to exchange */
)
```

**DESCRIPTION** This routine gets the specified number of bytes from *source*, exchanges the adjacent even and odd bytes, and puts them in *destination*.

---

**NOTE:** Due to speed considerations, this routine should only be used when absolutely necessary. Use **swab()** for aligned swaps.

---

It is an error for *nbytes* to be odd.

**RETURNS** N/A

**SEE ALSO** **bLib**, **swab()**

---

## **utime()**

**NAME** `utime()` – update time on a file

**SYNOPSIS**

```
int utime
(
    char *      file,
    struct utimbuf * newTimes
)
```

**DESCRIPTION**

**RETURNS** OK or ERROR.

**SEE ALSO** `dirLib`, `stat()`, `fstat()`, `ls()`

## va\_arg()

<b>NAME</b>	<b>va_arg()</b> – expand to an expression having the type and value of the call’s next argument
<b>SYNOPSIS</b>	<pre>void va_arg (     ap,                                /* list of type va_list */     type                               /* type */ )</pre>
<b>DESCRIPTION</b>	Each invocation of this macro modifies an object of type <b>va_list</b> ( <i>ap</i> ) so that the values of successive arguments are returned in turn. The parameter <i>type</i> is a type name specified such that the type of a pointer to an object that has the specified type can be obtained simply by postfixing a * to <i>type</i> . If there is no actual next argument, or if <i>type</i> is not compatible with the type of the actual next argument (as promoted according to the default argument promotions), the behavior is undefined.
<b>RETURNS</b>	The first invocation of <b>va_arg()</b> after <b>va_start()</b> returns the value of the argument after that specified by <i>parmN</i> (the rightmost parameter). Successive invocations return the value of the remaining arguments in succession.
<b>SEE ALSO</b>	<b>ansiStdarg</b>

---

## va\_end()

<b>NAME</b>	<b>va_end()</b> – facilitate a normal return from a routine using a <b>va_list</b> object
<b>SYNOPSIS</b>	<pre>void va_end (     ap                                /* list of type va_list */ )</pre>
<b>DESCRIPTION</b>	This macro facilitates a normal return from the function whose variable argument list was referred to by the expansion of <b>va_start()</b> that initialized the <b>va_list</b> object.  <b>va_end()</b> may modify the <b>va_list</b> object so that it is no longer usable (without an intervening invocation of <b>va_start()</b> ). If there is no corresponding invocation of the <b>va_start()</b> macro, or if the <b>va_end()</b> macro is not invoked before the return, the behavior is undefined.
<b>RETURNS</b>	N/A
<b>SEE ALSO</b>	<b>ansiStdarg</b>

---

## va\_start()

<b>NAME</b>	<code>va_start()</code> – initialize a <code>va_list</code> object for use by <code>va_arg()</code> and <code>va_end()</code>
<b>SYNOPSIS</b>	<pre>void va_start (     ap,                /* list of type va_list */     parmN              /* rightmost parameter */ )</pre>
<b>DESCRIPTION</b>	This macro initializes an object of type <code>va_list</code> ( <i>ap</i> ) for subsequent use by <code>va_arg()</code> and <code>va_end()</code> . The parameter <i>parmN</i> is the identifier of the rightmost parameter in the variable parameter list in the function definition (the one just before the, ...). If <i>parmN</i> is declared with the register storage class with a function or array type, or with a type that is not compatible with the type that results after application of the default argument promotions, the behavior is undefined.
<b>RETURNS</b>	N/A
<b>SEE ALSO</b>	<code>ansiStdarg</code>

---

## valloc()

<b>NAME</b>	<code>valloc()</code> – allocate memory on a page boundary
<b>SYNOPSIS</b>	<pre>void * valloc (     unsigned size     /* number of bytes to allocate */ )</pre>
<b>DESCRIPTION</b>	This routine allocates a buffer of <i>size</i> bytes from the system memory partition. Additionally, it insures that the allocated buffer begins on a page boundary. Page sizes are architecture-dependent.
<b>RETURNS</b>	A pointer to the newly allocated block, or <code>NULL</code> if the buffer could not be allocated or the memory management unit (MMU) support library has not been initialized.
<b>ERRNO</b>	<code>S_memLib_PAGE_SIZE_UNAVAILABLE</code>
<b>SEE ALSO</b>	<code>memLib</code>

## version()

- NAME** `version()` – print VxWorks version information
- SYNOPSIS** `void version (void)`
- DESCRIPTION** This command prints the VxWorks version number, the date this copy of VxWorks was made, and other pertinent information.
- EXAMPLE**
- ```
-> version
VxWorks (for Mizar 7170) version 5.1
Kernel: WIND version 2.1.
Made on Tue Jul 27 20:26:23 CDT 1997.
Boot line:
enp(0,0)host:/usr/wpwr/target/config/mz7170/vxWorks e=90.0.0.50 h=90.0.0.4
u=target
value = 1 = 0x1
```
- RETURNS** N/A
- SEE ALSO** `usrLib`, *VxWorks Programmer's Guide: Target Shell*, `windsh`, *Tornado User's Guide: Shell*
- 

## vfdprintf()

- NAME** `vfdprintf()` – write a string formatted with a variable argument list to a file descriptor
- SYNOPSIS**
- ```
int vfdprintf
(
    int          fd,          /* file descriptor to print to */
    const char * fmt,        /* format string for print */
    va_list      vaList     /* optional arguments to format */
)
```
- DESCRIPTION** This routine prints a string formatted with a variable argument list to a specified file descriptor. It is identical to `fdprintf()`, except that it takes the variable arguments to be formatted as a list *vaList* of type `va_list` rather than as in-line arguments.
- RETURNS** The number of characters output, or `ERROR` if there is an error during output.
- SEE ALSO** `fioLib`, `fdprintf()`

---

## fprintf()

<b>NAME</b>	<b>fprintf()</b> – write a formatted string to a stream (ANSI)
<b>SYNOPSIS</b>	<pre>int fprintf (     FILE *      fp,          /* stream to write to */     const char * fmt,       /* format string */     va_list     vaList      /* arguments to format string */ )</pre>
<b>DESCRIPTION</b>	This routine is equivalent to <b>fprintf()</b> , except that it takes the variable arguments to be formatted from a list <i>vaList</i> of type <b>va_list</b> rather than from in-line arguments.
<b>INCLUDE FILES</b>	<b>stdio.h</b>
<b>RETURNS</b>	The number of characters written, or a negative value if an output error occurs.
<b>SEE ALSO</b>	<b>ansiStdio, fprintf()</b>

---

## vmBaseGlobalMapInit()

<b>NAME</b>	<b>vmBaseGlobalMapInit()</b> – initialize global mapping
<b>SYNOPSIS</b>	<pre>VM_CONTEXT_ID vmBaseGlobalMapInit (     PHYS_MEM_DESC * pMemDescArray, /* pointer to array of mem desc */     int             numDescArrayElements, /* no. of elements in pMemDescArray */     BOOL            enable          /* enable virtual memory */ )</pre>
<b>DESCRIPTION</b>	<p>This routine creates and installs a virtual memory context with mappings defined for each contiguous memory segment defined in <i>pMemDescArray</i>. In the standard VxWorks configuration, an instance of <b>PHYS_MEM_DESC</b> (called <b>sysPhysMemDesc</b>) is defined in <b>sysLib.c</b>; the variable is passed to <b>vmBaseGlobalMapInit()</b> by the system configuration mechanism.</p> <p>The physical memory descriptor also contains state information used to initialize the state information in the MMU's translation table for that memory segment. The following state bits may be or'ed together:</p>

VM_STATE_VALID	VM_STATE_VALID_NOT	valid/invalid
VM_STATE_WRITABLE	VM_STATE_WRITABLE_NOT	writable/write-protected
VM_STATE_CACHEABLE	VM_STATE_CACHEABLE_NOT	cacheable/not-cacheable

Additionally, mask bits are or'ed together in the **initialStateMask** structure element to describe which state bits are being specified in the **initialState** structure element:

```
VM_STATE_MASK_VALID
VM_STATE_MASK_WRITABLE
VM_STATE_MASK_CACHEABLE
```

If *enable* is TRUE, the MMU is enabled upon return.

**RETURNS** A pointer to a newly created virtual memory context, or NULL if memory cannot be mapped.

**SEE ALSO** [vmBaseLib](#), [vmBaseLibInit\(\)](#)

---

## vmBaseLibInit()

**NAME** [vmBaseLibInit\(\)](#) – initialize base virtual memory support

**SYNOPSIS**

```
STATUS vmBaseLibInit
(
    int pageSize           /* size of page */
)
```

**DESCRIPTION** This routine initializes the virtual memory context class and module-specific data structures. It is called only once during system initialization, and should be followed with a call to [vmBaseGlobalMapInit\(\)](#), which initializes and enables the MMU.

**RETURNS** OK.

**SEE ALSO** [vmBaseLib](#), [vmBaseGlobalMapInit\(\)](#)

---

## vmBasePageSizeGet()

**NAME** `vmBasePageSizeGet()` – return the page size

**SYNOPSIS** `int vmBasePageSizeGet (void)`

**DESCRIPTION** This routine returns the architecture-dependent page size.  
 This routine is callable from interrupt level.

**RETURNS** The page size of the current architecture.

**SEE ALSO** `vmBaseLib`

---

## vmBaseStateSet()

**NAME** `vmBaseStateSet()` – change the state of a block of virtual memory

**SYNOPSIS** `STATUS vmBaseStateSet`

```
(
    VM_CONTEXT_ID context,    /* context - NULL == currentContext */
    void *        pVirtual,  /* virtual address to modify state of */
    int           len,       /* len of virtual space to modify state of */
    UINT          stateMask, /* state mask */
    UINT          state      /* state */
)
```

**DESCRIPTION** This routine changes the state of a block of virtual memory. Each page of virtual memory has at least three elements of state information: validity, writability, and cacheability. Specific architectures may define additional state information; see `vmLib.h` for additional architecture-specific states. Memory accesses to a page marked as invalid will result in an exception. Pages may be invalidated to prevent them from being corrupted by invalid references. Pages may be defined as read-only or writable, depending on the state of the writable bits. Memory accesses to pages marked as not-cacheable will always result in a memory cycle, bypassing the cache. This is useful for multiprocessing, multiple bus masters, and hardware control registers.

The following states are provided and may be or'ed together in the state parameter:

<code>VM_STATE_VALID</code>	<code>VM_STATE_VALID_NOT</code>	valid/invalid
<code>VM_STATE_WRITABLE</code>	<code>VM_STATE_WRITABLE_NOT</code>	writable/write-protected
<code>VM_STATE_CACHEABLE</code>	<code>VM_STATE_CACHEABLE_NOT</code>	cacheable/not-cacheable

Additionally, the following masks are provided so that only specific states may be set. These may be or'ed together in the **stateMask** parameter.

**VM\_STATE\_MASK\_VALID**  
**VM\_STATE\_MASK\_WRITABLE**  
**VM\_STATE\_MASK\_CACHEABLE**

If *context* is specified as **NULL**, the current context is used.

This routine is callable from interrupt level.

- RETURNS** OK, or **ERROR** if the validation fails, *pVirtual* is not on a page boundary, *len* is not a multiple of the page size, or the architecture-dependent state set fails for the specified virtual address.
- ERRNO** **S\_vmLib\_NOT\_PAGE\_ALIGNED**, **S\_vmLib\_BAD\_STATE\_PARAM**,  
**S\_vmLib\_BAD\_MASK\_PARAM**
- SEE ALSO** **vmBaseLib**

---

## vmContextCreate()

- NAME** **vmContextCreate()** – create a new virtual memory context (VxVMI Opt.)
- SYNOPSIS** **VM\_CONTEXT\_ID vmContextCreate (void)**
- DESCRIPTION** This routine creates a new virtual memory context. The newly created context does not become the current context until explicitly installed by a call to **vmCurrentSet()**. Modifications to the context state (mappings, state changes, etc.) may be performed on any virtual memory context, even if it is not the current context.
- This routine should not be called from interrupt level.
- AVAILABILITY** This routine is distributed as a component of the unbundled virtual memory support option, VxVMI.
- RETURNS** A pointer to a new virtual memory context, or **NULL** if the allocation or initialization fails.
- SEE ALSO** **vmLib**

---

## vmContextDelete()

<b>NAME</b>	<b>vmContextDelete()</b> – delete a virtual memory context (VxVMI Opt.)
<b>SYNOPSIS</b>	<pre><b>STATUS</b> vmContextDelete (     <b>VM_CONTEXT_ID</b> context )</pre>
<b>DESCRIPTION</b>	<p>This routine deallocates the underlying translation table associated with a virtual memory context. It does not free the physical memory already mapped into the virtual memory space.</p> <p>This routine should not be called from interrupt level.</p>
<b>AVAILABILITY</b>	This routine is distributed as a component of the unbundled virtual memory support option, VxVMI.
<b>RETURNS</b>	OK, or <b>ERROR</b> if <i>context</i> is not a valid context descriptor or if an error occurs deleting the translation table.
<b>SEE ALSO</b>	<b>vmLib</b>

---

## vmContextShow()

<b>NAME</b>	<b>vmContextShow()</b> – display the translation table for a context (VxVMI Opt.)
<b>SYNOPSIS</b>	<pre><b>STATUS</b> vmContextShow (     <b>VM_CONTEXT_ID</b> context    /* context - NULL == currentContext */ )</pre>
<b>DESCRIPTION</b>	<p>This routine displays the translation table for a specified context. If <i>context</i> is specified as <b>NULL</b>, the current context is displayed. Output is formatted to show blocks of virtual memory with consecutive physical addresses and the same state. State information shows the writable and cacheable states. If the block is in global virtual memory, the word “global” is appended to the line. Only virtual memory that has its valid state bit set is displayed.</p> <p>This routine should be used for debugging purposes only.</p> <p>Note that this routine cannot report non-standard architecture-dependent states.</p>

<b>AVAILABILITY</b>	This routine is distributed as a component of the unbundled virtual memory support option, VxVMI.
<b>RETURNS</b>	OK, or ERROR if the virtual memory context is invalid.
<b>SEE ALSO</b>	vmShow

---

## vmCurrentGet()

<b>NAME</b>	vmCurrentGet() – get the current virtual memory context (VxVMI Opt.)
<b>SYNOPSIS</b>	<code>VM_CONTEXT_ID vmCurrentGet (void)</code>
<b>DESCRIPTION</b>	This routine returns the current virtual memory context. This routine is callable from interrupt level.
<b>AVAILABILITY</b>	This routine is distributed as a component of the unbundled virtual memory support option, VxVMI.
<b>RETURNS</b>	The current virtual memory context, or NULL if no virtual memory context is installed.
<b>SEE ALSO</b>	vmLib

---

## vmCurrentSet()

<b>NAME</b>	vmCurrentSet() – set the current virtual memory context (VxVMI Opt.)
<b>SYNOPSIS</b>	<pre>STATUS vmCurrentSet (     VM_CONTEXT_ID context    /* context to install */ )</pre>
<b>DESCRIPTION</b>	This routine installs a specified virtual memory context. This routine is callable from interrupt level.
<b>AVAILABILITY</b>	This routine is distributed as a component of the unbundled virtual memory support option, VxVMI.
<b>RETURNS</b>	OK, or ERROR if the validation or context switch fails.
<b>SEE ALSO</b>	vmLib

---

## vmEnable()

<b>NAME</b>	<b>vmEnable()</b> – enable or disable virtual memory (VxVMI Opt.)
<b>SYNOPSIS</b>	<pre>STATUS vmEnable (     BOOL enable          /* TRUE == enable MMU, FALSE == disable MMU */ )</pre>
<b>DESCRIPTION</b>	<p>This routine turns virtual memory on and off. Memory management should not be turned off once it is turned on except in the case of system shutdown.</p> <p>This routine is callable from interrupt level.</p>
<b>AVAILABILITY</b>	This routine is distributed as a component of the unbundled virtual memory support option, VxVMI.
<b>RETURNS</b>	OK, or <b>ERROR</b> if the validation or architecture-dependent code fails.
<b>SEE ALSO</b>	<b>vmLib</b>

---

## vmGlobalInfoGet()

<b>NAME</b>	<b>vmGlobalInfoGet()</b> – get global virtual memory information (VxVMI Opt.)
<b>SYNOPSIS</b>	<pre>UINT8 *vmGlobalInfoGet (void)</pre>
<b>DESCRIPTION</b>	<p>This routine provides a description of those parts of the virtual memory space dedicated to global memory. The routine returns a pointer to an array of <b>UINT8</b>. Each element of the array corresponds to a block of virtual memory, the size of which is architecture-dependent and can be obtained with a call to <b>vmPageBlockSizeGet()</b>. To determine if a particular address is in global virtual memory, use the following code:</p> <pre>UINT8 *globalPageBlockArray = vmGlobalInfoGet (); int pageBlockSize = vmPageBlockSizeGet ();  if (globalPageBlockArray[addr/pageBlockSize])     ...</pre>

The array pointed to by the returned pointer is guaranteed to be static as long as no calls are made to **vmGlobalMap()** while the array is being examined. The information in the

array can be used to determine what portions of the virtual memory space are available for use as private virtual memory within a virtual memory context.

This routine is callable from interrupt level.

<b>AVAILABILITY</b>	This routine is distributed as a component of the unbundled virtual memory support option, VxVMI.
<b>RETURNS</b>	A pointer to an array of UINT8.
<b>SEE ALSO</b>	<b>vmLib</b> , <b>vmPageBlockSizeGet()</b>

---

## vmGlobalMap()

**NAME** **vmGlobalMap()** – map physical pages to virtual space in shared global virtual memory (VxVMI Opt.)

**SYNOPSIS**

```
STATUS vmGlobalMap
(
    void * virtualAddr,      /* virtual address */
    void * physicalAddr,    /* physical address */
    UINT  len                /* len of virtual and physical spaces */
)
```

**DESCRIPTION** This routine maps physical pages to virtual space that is shared by all virtual memory contexts. Calls to **vmGlobalMap()** should be made before any virtual memory contexts are created to insure that the shared global mappings are included in all virtual memory contexts. Mappings created with **vmGlobalMap()** after virtual memory contexts are created are not guaranteed to appear in all virtual memory contexts. After the call to **vmGlobalMap()**, the state of all pages in the newly mapped virtual memory is unspecified and must be set with a call to **vmStateSet()**, once the initial virtual memory context is created.

This routine should not be called from interrupt level.

<b>AVAILABILITY</b>	This routine is distributed as a component of the unbundled virtual memory support option, VxVMI.
<b>RETURNS</b>	<b>OK</b> , or <b>ERROR</b> if <i>virtualAddr</i> or <i>physicalAddr</i> are not on page boundaries, <i>len</i> is not a multiple of the page size, or the mapping fails.
<b>ERRNO</b>	<b>S_vmLib_NOT_PAGE_ALIGNED</b>
<b>SEE ALSO</b>	<b>vmLib</b>

---

## vmGlobalMapInit()

**NAME** `vmGlobalMapInit()` – initialize global mapping (VxVMI Opt.)

**SYNOPSIS**

```
VM_CONTEXT_ID vmGlobalMapInit
(
    PHYS_MEM_DESC * pMemDescArray, /* pointer to array of mem descs */
    int          numDescArrayElements, /* num of elements in pMemDescArray */
    BOOL         enable                /* enable virtual memory */
)
```

**DESCRIPTION** This routine is a convenience routine that creates and installs a virtual memory context with global mappings defined for each contiguous memory segment defined in the physical memory descriptor array passed as an argument. The context ID returned becomes the current virtual memory context.

The physical memory descriptor also contains state information used to initialize the state information in the MMU's translation table for that memory segment. The following state bits may be or'ed together:

VM_STATE_VALID	VM_STATE_VALID_NOT	valid/invalid
VM_STATE_WRITABLE	VM_STATE_WRITABLE_NOT	writable/write-protected
VM_STATE_CACHEABLE	VM_STATE_CACHEABLE_NOT	cacheable/not-cacheable

Additionally, mask bits are or'ed together in the **initialStateMask** structure element to describe which state bits are being specified in the **initialState** structure element:

```
VM_STATE_MASK_VALID
VM_STATE_MASK_WRITABLE
VM_STATE_MASK_CACHEABLE
```

If the *enable* parameter is **TRUE**, the MMU is enabled upon return. The **vmGlobalMapInit()** routine should be called only after **vmLibInit()** has been called.

**AVAILABILITY** This routine is distributed as a component of the unbundled virtual memory support option, VxVMI.

**RETURNS** A pointer to a newly created virtual memory context, or **NULL** if the memory cannot be mapped.

**SEE ALSO** `vmLib`

## vmLibInit()

<b>NAME</b>	<b>vmLibInit()</b> – initialize the virtual memory support module (VxVMI Opt.)
<b>SYNOPSIS</b>	<pre>STATUS vmLibInit (     int pageSize          /* size of page */ )</pre>
<b>DESCRIPTION</b>	This routine initializes the virtual memory context class. It is called only once during system initialization.
<b>AVAILABILITY</b>	This routine is distributed as a component of the unbundled virtual memory support option, VxVMI.
<b>RETURNS</b>	OK.
<b>SEE ALSO</b>	vmLib

---

## vmMap()

<b>NAME</b>	<b>vmMap()</b> – map physical space into virtual space (VxVMI Opt.)
<b>SYNOPSIS</b>	<pre>STATUS vmMap (     VM_CONTEXT_ID context,      /* context - NULL == currentContext */     void *      virtualAddr, /* virtual address */     void *      physicalAddr, /* physical address */     UINT      len          /* len of virtual and physical spaces */ )</pre>
<b>DESCRIPTION</b>	<p>This routine maps physical pages into a contiguous block of virtual memory. <i>virtualAddr</i> and <i>physicalAddr</i> must be on page boundaries, and <i>len</i> must be evenly divisible by the page size. After the call to <b>vmMap()</b>, the state of all pages in the newly mapped virtual memory is valid, writable, and cacheable.</p> <p>The <b>vmMap()</b> routine can fail if the specified virtual address space conflicts with the translation tables of the global virtual memory space. The global virtual address space is architecture-dependent and is initialized at boot time with calls to <b>vmGlobalMap()</b> by <b>vmGlobalMapInit()</b>. If a conflict results, <b>errno</b> is set to</p>

**S\_vmLib\_ADDR\_IN\_GLOBAL\_SPACE**. To avoid this conflict, use **vmGlobalInfoGet()** to ascertain which portions of the virtual address space are reserved for the global virtual address space. If *context* is specified as **NULL**, the current virtual memory context is used.

This routine should not be called from interrupt level.

<b>AVAILABILITY</b>	This routine is distributed as a component of the unbundled virtual memory support option, VxVMI.
<b>RETURNS</b>	<b>OK</b> , or <b>ERROR</b> if <i>virtualAddr</i> or <i>physicalAddr</i> are not on page boundaries, <i>len</i> is not a multiple of the page size, the validation fails, or the mapping fails.
<b>ERRNO</b>	<b>S_vmLib_NOT_PAGE_ALIGNED</b> , <b>S_vmLib_ADDR_IN_GLOBAL_SPACE</b>
<b>SEE ALSO</b>	<b>vmLib</b>

---

## vmPageBlockSizeGet()

<b>NAME</b>	<b>vmPageBlockSizeGet()</b> – get the architecture-dependent page block size (VxVMI Opt.)
<b>SYNOPSIS</b>	<b>int vmPageBlockSizeGet (void)</b>
<b>DESCRIPTION</b>	<p>This routine returns the size of a page block for the current architecture. Each MMU architecture constructs translation tables such that a minimum number of pages are pre-defined when a new section of the translation table is built. This minimal group of pages is referred to as a “page block.” This routine may be used in conjunction with <b>vmGlobalInfoGet()</b> to examine the layout of global virtual memory.</p> <p>This routine is callable from interrupt level.</p>
<b>AVAILABILITY</b>	This routine is distributed as a component of the unbundled virtual memory support option, VxVMI.
<b>RETURNS</b>	The page block size of the current architecture.
<b>SEE ALSO</b>	<b>vmLib</b> , <b>vmGlobalInfoGet()</b>

## vmPageSizeGet( )

<b>NAME</b>	<b>vmPageSizeGet( )</b> – return the page size (VxVMI Opt.)
<b>SYNOPSIS</b>	<code>int vmPageSizeGet (void)</code>
<b>DESCRIPTION</b>	This routine returns the architecture-dependent page size. This routine is callable from interrupt level.
<b>AVAILABILITY</b>	This routine is distributed as a component of the unbundled virtual memory support option, VxVMI.
<b>RETURNS</b>	The page size of the current architecture.
<b>SEE ALSO</b>	<b>vmLib</b>

---

## vmShowInit( )

<b>NAME</b>	<b>vmShowInit( )</b> – include virtual memory show facility (VxVMI Opt.)
<b>SYNOPSIS</b>	<code>void vmShowInit (void)</code>
<b>DESCRIPTION</b>	This routine acts as a hook to include <b>vmContextShow( )</b> . It is called automatically when the virtual memory show facility is configured into VxWorks using either of the following methods: <ul style="list-style-type: none"><li>– If you use the configuration header files, define both <b>INCLUDE_MMU_FULL</b> and <b>INCLUDE_SHOW_ROUTINES</b> in <b>config.h</b>.</li><li>– If you use the Tornado project facility, select <b>INCLUDE_MMU_FULL_SHOW</b>.</li></ul>
<b>AVAILABILITY</b>	* This routine is distributed as a component of the unbundled virtual memory support option, VxVMI.
<b>RETURNS</b>	N/A
<b>SEE ALSO</b>	<b>vmShow</b>

---

## vmStateGet()

<b>NAME</b>	<b>vmStateGet()</b> – get the state of a page of virtual memory (VxVMI Opt.)
<b>SYNOPSIS</b>	<pre>STATUS vmStateGet (     VM_CONTEXT_ID context,    /* context - NULL == currentContext */     void *          pPageAddr, /* virtual page addr */     UINT *          pState     /* where to return state */ )</pre>
<b>DESCRIPTION</b>	<p>This routine extracts state bits with the following masks:</p> <pre>VM_STATE_MASK_VALID VM_STATE_MASK_WRITABLE VM_STATE_MASK_CACHEABLE</pre> <p>Individual states may be identified with the following constants:</p> <pre>VM_STATE_VALID      VM_STATE_VALID_NOT    valid/invalid VM_STATE_WRITABLE   VM_STATE_WRITABLE_NOT  writable/write-protected VM_STATE_CACHEABLE  VM_STATE_CACHEABLE_NOT cacheable/not-cacheable</pre> <p>For example, to see if a page is writable, the following code would be used:</p> <pre>vmStateGet (vmContext, pageAddr, &amp;state); if ((state &amp; VM_STATE_MASK_WRITABLE) &amp; VM_STATE_WRITABLE)     ...</pre> <p>If <i>context</i> is specified as <b>NULL</b>, the current virtual memory context is used.</p> <p>This routine is callable from interrupt level.</p>
<b>AVAILABILITY</b>	This routine is distributed as a component of the unbundled virtual memory support option, VxVMI.
<b>RETURNS</b>	<b>OK</b> , or <b>ERROR</b> if <i>pageAddr</i> is not on a page boundary, the validity check fails, or the architecture-dependent state get fails for the specified virtual address.
<b>ERRNO</b>	<b>S_vmLib_NOT_PAGE_ALIGNED</b>
<b>SEE ALSO</b>	<b>vmLib</b>

---

## vmStateSet()

**NAME** `vmStateSet()` – change the state of a block of virtual memory (VxVMI Opt.)

**SYNOPSIS**

```
STATUS vmStateSet
(
    VM_CONTEXT_ID context,    /* context - NULL == currentContext */
    void *          pVirtual, /* virtual address to modify state of */
    int             len,      /* len of virtual space to modify state of */
    UINT           stateMask, /* state mask */
    UINT           state      /* state */
)
```

**DESCRIPTION** This routine changes the state of a block of virtual memory. Each page of virtual memory has at least three elements of state information: validity, writability, and cacheability. Specific architectures may define additional state information; see **vmLib.h** for additional architecture-specific states. Memory accesses to a page marked as invalid will result in an exception. Pages may be invalidated to prevent them from being corrupted by invalid references. Pages may be defined as read-only or writable, depending on the state of the writable bits. Memory accesses to pages marked as not-cacheable will always result in a memory cycle, bypassing the cache. This is useful for multiprocessing, multiple bus masters, and hardware control registers.

The following states are provided and may be or'ed together in the state parameter:

VM_STATE_VALID	VM_STATE_VALID_NOT	valid/invalid
VM_STATE_WRITABLE	VM_STATE_WRITABLE_NOT	writable/write-protected
VM_STATE_CACHEABLE	VM_STATE_CACHEABLE_NOT	cacheable/not-cacheable

Additionally, the following masks are provided so that only specific states may be set. These may be or'ed together in the **stateMask** parameter.

```
VM_STATE_MASK_VALID
VM_STATE_MASK_WRITABLE
VM_STATE_MASK_CACHEABLE
```

If *context* is specified as **NULL**, the current context is used.

This routine is callable from interrupt level.

**AVAILABILITY** This routine is distributed as a component of the unbundled virtual memory support option, VxVMI.

**RETURNS** **OK** or **ERROR** if the validation fails, *pVirtual* is not on a page boundary, *len* is not a multiple of page size, or the architecture-dependent state set fails for the specified virtual address.

**ERRNO** S\_vmLib\_NOT\_PAGE\_ALIGNED, S\_vmLib\_BAD\_STATE\_PARAM,  
S\_vmLib\_BAD\_MASK\_PARAM

**SEE ALSO** vmLib

---

## vmTextProtect()

**NAME** vmTextProtect() – write-protect a text segment (VxVMI Opt.)

**SYNOPSIS** **STATUS** vmTextProtect (void)

**DESCRIPTION** This routine write-protects the VxWorks text segment and sets a flag so that all text segments loaded by the incremental loader will be write-protected. The routine should be called after both **vmLibInit()** and **vmGlobalMapInit()** have been called.

**AVAILABILITY** This routine is distributed as a component of the unbundled virtual memory support option, VxVMI.

**RETURNS** OK, or ERROR if the text segment cannot be write-protected.

**ERRNO** S\_vmLib\_TEXT\_PROTECTION\_UNAVAILABLE

**SEE ALSO** vmLib

---

## vmTranslate()

**NAME** vmTranslate() – translate a virtual address to a physical address (VxVMI Opt.)

**SYNOPSIS** **STATUS** vmTranslate

```
(
    VM_CONTEXT_ID context,      /* context - NULL == currentContext */
    void *        virtualAddr, /* virtual address */
    void *        *physicalAddr /* place to put result */
)
```

**DESCRIPTION** This routine retrieves mapping information for a virtual address from the page translation tables. If the specified virtual address has never been mapped, the returned status can be either OK or ERROR; however, if it is OK, then the returned physical address will be -1. If *context* is specified as NULL, the current context is used.

This routine is callable from interrupt level.

- AVAILABILITY** This routine is distributed as a component of the unbundled virtual memory support option, VxVML.
- RETURNS** OK, or ERROR if the validation or translation fails.
- SEE ALSO** vmLib

---

## vprintf()

**NAME** vprintf() – write a string formatted with a variable argument list to standard output (ANSI)

**SYNOPSIS**

```
int vprintf
(
    const char * fmt,          /* format string to write */
    va_list     vaList        /* arguments to format */
)
```

**DESCRIPTION** This routine prints a string formatted with a variable argument list to standard output. It is identical to **printf()**, except that it takes the variable arguments to be formatted as a list *vaList* of type **va\_list** rather than as in-line arguments.

**RETURNS** The number of characters output, or ERROR if there is an error during output.

**SEE ALSO** **fcntlLib**, **printf()**, *American National Standard for Information Systems -Programming Language - C, ANSI X3.159-1989: Input/Output (stdio.h)*

---

## vsprintf()

**NAME** vsprintf() – write a string formatted with a variable argument list to a buffer (ANSI)

**SYNOPSIS**

```
int vsprintf
(
    char *      buffer,        /* buffer to write to */
    const char * fmt,          /* format string */
    va_list     vaList        /* optional arguments to format */
)
```

- DESCRIPTION** This routine copies a string formatted with a variable argument list to a specified buffer. This routine is identical to **sprintf()**, except that it takes the variable arguments to be formatted as a list *vaList* of type **va\_list** rather than as in-line arguments.
- RETURNS** The number of characters copied to *buffer*, not including the NULL terminator.
- SEE ALSO** **fxLib**, **sprintf()**, *American National Standard for Information Systems -Programming Language - C, ANSI X3.159-1989: Input/Output (stdio.h)*

---

## vxCr2Get()

- NAME** **vxCr2Get()** – get a content of the Control Register 2 (x86)
- SYNOPSIS** **int vxCr2Get (void)**
- DESCRIPTION** This routine gets a content of the Control Register 2.
- RETURNS** a value of the Control Register 2
- SEE ALSO** **vxLib**

---

## vxCr2Set()

- NAME** **vxCr2Set()** – set a value to the Control Register 2 (x86)
- SYNOPSIS** **void vxCr2Set**  
(  
    **int value**                    /\* CR2 value \*/  
)
- DESCRIPTION** This routine sets a value to the Control Register 2.
- RETURNS** N/A
- SEE ALSO** **vxLib**

## **vxCr3Get()**

<b>NAME</b>	<b>vxCr3Get()</b> – get a content of the Control Register 3 (x86)
<b>SYNOPSIS</b>	<code>int vxCr3Get (void)</code>
<b>DESCRIPTION</b>	This routine gets a content of the Control Register 3.
<b>RETURNS</b>	a value of the Control Register 3
<b>SEE ALSO</b>	<b>vxLib</b>

---

## **vxCr3Set()**

<b>NAME</b>	<b>vxCr3Set()</b> – set a value to the Control Register 3 (x86)
<b>SYNOPSIS</b>	<pre>void vxCr3Set (     int value          /* CR3 value */ )</pre>
<b>DESCRIPTION</b>	This routine sets a value to the Control Register 3.
<b>RETURNS</b>	N/A
<b>SEE ALSO</b>	<b>vxLib</b>

---

## **vxCr4Get()**

<b>NAME</b>	<b>vxCr4Get()</b> – get a content of the Control Register 4 (x86)
<b>SYNOPSIS</b>	<code>int vxCr4Get (void)</code>
<b>DESCRIPTION</b>	This routine gets a content of the Control Register 4.
<b>RETURNS</b>	a value of the Control Register 4
<b>SEE ALSO</b>	<b>vxLib</b>

---

## vxCr4Set()

**NAME** vxCr4Set() – set a value to the Control Register 4 (x86)

**SYNOPSIS**

```
void vxCr4Set  
(  
    int value          /* CR4 value */  
)
```

**DESCRIPTION** This routine sets a value to the Control Register 4.

**RETURNS** N/A

**SEE ALSO** vxLib

---

## vxCr0Get()

**NAME** vxCr0Get() – get a content of the Control Register 0 (x86)

**SYNOPSIS**

```
int vxCr0Get (void)
```

**DESCRIPTION** This routine gets a content of the Control Register 0.

**RETURNS** a value of the Control Register 0

**SEE ALSO** vxLib

## **vxCr0Set()**

**NAME** vxCr0Set() – set a value to the Control Register 0 (x86)

**SYNOPSIS**

```
void vxCr0Set
(
    int value          /* CR0 value */
)
```

**DESCRIPTION** This routine sets a value to the Control Register 0.

**RETURNS** N/A

**SEE ALSO** vxLib

---

## **vxDrGet()**

**NAME** vxDrGet() – get a content of the Debug Register 0 to 7 (x86)

**SYNOPSIS**

```
void vxDrGet
(
    int * pDr0,        /* DR0 */
    int * pDr1,        /* DR1 */
    int * pDr2,        /* DR2 */
    int * pDr3,        /* DR3 */
    int * pDr4,        /* DR4 */
    int * pDr5,        /* DR5 */
    int * pDr6,        /* DR6 */
    int * pDr7         /* DR7 */
)
```

**DESCRIPTION** This routine gets a content of the Debug Register 0 to 7.

**RETURNS** N/A

**SEE ALSO** vxLib

---

## vxDrSet()

**NAME** vxDrSet() – set a value to the Debug Register 0 to 7 (x86)

**SYNOPSIS**

```
void vxDrSet
(
    int dr0,          /* DR0 */
    int dr1,          /* DR1 */
    int dr2,          /* DR2 */
    int dr3,          /* DR3 */
    int dr4,          /* DR4 */
    int dr5,          /* DR5 */
    int dr6,          /* DR6 */
    int dr7           /* DR7 */
)
```

**DESCRIPTION** This routine sets a value to the Debug Register 0 to 7.

**RETURNS** N/A

**SEE ALSO** vxLib

---

## vxEflagsGet()

**NAME** vxEflagsGet() – get a content of the EFLAGS register (x86)

**SYNOPSIS** int vxEflagsGet (void)

**DESCRIPTION** This routine gets a content of the EFLAGS register

**RETURNS** a value of the EFLAGS register

**SEE ALSO** vxLib



## **vxEflagsSet()**

**NAME** vxEflagsSet() – set a value to the EFLAGS register (x86)

**SYNOPSIS**

```
void vxEflagsSet
(
    int value          /* EFLAGS value */
)
```

**DESCRIPTION** This routine sets a value to the EFLAGS register

**RETURNS** N/A

**SEE ALSO** vxLib

---

## **vxGdtrGet()**

**NAME** vxGdtrGet() – get a content of the Global Descriptor Table Register (x86)

**SYNOPSIS**

```
void vxGdtrGet
(
    long long int * pGdtr /* memory to store GDTR */
)
```

**DESCRIPTION** This routine gets a content of the Global Descriptor Table Register

**RETURNS** N/A

**SEE ALSO** vxLib

---

## vxIdtrGet()

**NAME** vxIdtrGet() – get a content of the Interrupt Descriptor Table Register (x86)

**SYNOPSIS**

```
void vxIdtrGet
(
    long long int * pIdtr    /* memory to store IDTR */
)
```

**DESCRIPTION** This routine gets a content of the Interrupt Descriptor Table Register

**RETURNS** N/A

**SEE ALSO** vxLib

---

## vxLdtrGet()

**NAME** vxLdtrGet() – get a content of the Local Descriptor Table Register (x86)

**SYNOPSIS**

```
void vxLdtrGet
(
    long long int * pLdtr    /* memory to store LDTR */
)
```

**DESCRIPTION** This routine gets a content of the Local Descriptor Table Register

**RETURNS** N/A

**SEE ALSO** vxLib

## vxMemArchProbe()

**NAME** vxMemArchProbe() – architecture-specific part of vxMemProbe()

**SYNOPSIS**

```
STATUS vxMemArchProbe
(
    char * adrs,          /* address to be probed */
    int   mode,          /* VX_READ or VX_WRITE */
    int   length,        /* 1, 2, 4, or 8 */
    char * pVal          /* where to return value, or ptr to value */
                        /* to be written */
)
```

**DESCRIPTION** This is the routine implementing the architecture specific part of the vxMemProbe() routine. It traps the relevant exceptions while accessing the specified address. If an exception occurs, it returns **ERROR**. If no exception occurs, it returns **OK**.

**RETURNS** OK or **ERROR** if an exception occurred during access.

**SEE ALSO** vxLib

---

## vxMemProbe()

**NAME** vxMemProbe() – probe an address for a bus error

**SYNOPSIS**

```
STATUS vxMemProbe
(
    char * adrs,          /* address to be probed */
    int   mode,          /* VX_READ or VX_WRITE */
    int   length,        /* 1, 2, 4, or 8 */
    char * pVal          /* where to return value, or ptr to value */
                        /* to be written */
)
```

**DESCRIPTION** This routine probes a specified address to see if it is readable or writable, as specified by *mode*. The address is read or written as 1, 2, or 4 bytes, as specified by *length* (values other than 1, 2, or 4 yield unpredictable results). If the probe is a **VX\_READ** (0), the value read is copied to the location pointed to by *pVal*. If the probe is a **VX\_WRITE** (1), the value written

is taken from the location pointed to by *pVal*. In either case, *pVal* should point to a value of 1, 2, or 4 bytes, as specified by *length*.

Note that only bus errors are trapped during the probe, and that the access must otherwise be valid (*i.e.*, it must not generate an address error).

```

EXAMPLE      testMem (adrs)
              char *adrs;
              {
              char testW = 1;
              char testR;
              if (vxMemProbe (adrs, VX_WRITE, 1, &testW) == OK)
                  printf ("value %d written to adrs %x\n", testW, adrs);
              if (vxMemProbe (adrs, VX_READ, 1, &testR) == OK)
                  printf ("value %d read from adrs %x\n", testR, adrs);
              }

```

**MODIFICATION** The BSP can modify the behavior of **vxMemProbe()** by supplying an alternate routine and placing the address in the global variable **\_func\_vxMemProbeHook**. The BSP routine will be called instead of the architecture specific routine **vxMemArchProbe()**.

**RETURNS** OK, or **ERROR** if the probe caused a bus error or was misaligned.

**SEE ALSO** vxLib, vxMemArchProbe()

---

## vxPowerDown()

**NAME** vxPowerDown() – place the processor in reduced-power mode (PowerPC, SH)

**SYNOPSIS** UINT32 vxPowerDown (void)

**DESCRIPTION** This routine activates the reduced-power mode if power management is enabled. It is called by the scheduler when the kernel enters the idle loop. The power management mode is selected by **vxPowerModeSet()**.

**RETURNS** OK, or **ERROR** if power management is not supported or if external interrupts are disabled.

**SEE ALSO** vxLib, vxPowerModeSet(), vxPowerModeGet()

---

## vxPowerModeGet()

<b>NAME</b>	<b>vxPowerModeGet()</b> – get the power management mode (PowerPC, SH, x86)
<b>SYNOPSIS</b>	<b>UINT32 vxPowerModeGet (void)</b>
<b>DESCRIPTION</b>	This routine returns the power management mode set by <b>vxPowerModeSet()</b> .
<b>RETURNS</b>	The power management mode, or <b>ERROR</b> if no mode has been selected or if power management is not supported.
<b>SEE ALSO</b>	<b>vxLib</b> , <b>vxPowerModeSet()</b> , <b>vxPowerDown()</b>

---

## vxPowerModeSet()

<b>NAME</b>	<b>vxPowerModeSet()</b> – set the power management mode (PowerPC, SH, x86)
<b>SYNOPSIS</b>	<b>STATUS vxPowerModeSet</b> ( <b>UINT32 mode</b> /* power management mode to select */ )
<b>DESCRIPTION</b>	This routine selects the power management mode to be activated when <b>vxPowerDown()</b> is called. <b>vxPowerModeSet()</b> is normally called in the BSP initialization routine <b>sysHwInit()</b> .
<b>USAGE PPC</b>	Power management modes include the following: <b>VX_POWER_MODE_DISABLE</b> (0x1) Power management is disabled; this prevents the MSR(POW) bit from being set (all PPC). <b>VX_POWER_MODE_FULL</b> (0x2) All CPU units are active while the kernel is idle (PPC603, PPCEC603 and PPC860 only). <b>VX_POWER_MODE_DOZE</b> (0x4) Only the decremter, data cache, and bus snooping are active while the kernel is idle (PPC603, PPCEC603 and PPC860). <b>VX_POWER_MODE_NAP</b> (0x8) Only the decremter is active while the kernel is idle (PPC603, PPCEC603 and PPC604).

**VX\_POWER\_MODE\_SLEEP** (0x10)

All CPU units are inactive while the kernel is idle (PPC603, PPCEC603 and PPC860 - not recommended for the PPC603 and PPCEC603 architecture).

**VX\_POWER\_MODE\_DEEP\_SLEEP** (0x20)

All CPU units are inactive while the kernel is idle (PPC860 only - not recommended).

**VX\_POWER\_MODE\_DPM** (0x40)

Dynamic Power Management Mode (PPC603 and PPCEC603 only).

**VX\_POWER\_MODE\_DOWN** (0x80)

Only a hard reset causes an exit from power-down low power mode (PPC860 only - not recommended).

**USAGE SH**

Power management modes include the following:

**VX\_POWER\_MODE\_DISABLE** (0x0)

Power management is disabled.

**VX\_POWER\_MODE\_SLEEP** (0x1)

The core CPU is halted, on-chip peripherals operating, external memory refreshing.

**VX\_POWER\_MODE\_DEEP\_SLEEP** (0x2)

The core CPU is halted, on-chip peripherals operating, external memory self-refreshing (SH-4 only).

**VX\_POWER\_MODE\_USER** (0xff)

Set up to three 8-bit standby registers with user-specified values:

```
vxPowerModeSet (VX_POWER_MODE_USER | sbr1<<8 | sbr2<<16 | sbr3<<24);
```

The sbr1 value is written to the STBCR or SBYCR1, sbr2 is written to the STBCR2 or SBYCR2, and sbr3 is written to the STBCR3 register (when available), depending on the SH processor type.

USAGE x86: **vxPowerModeSet()** is called in the BSP initialization routine **sysHwInit()**. Power management modes include the following:

**VX\_POWER\_MODE\_DISABLE** (0x1)

Power management is disable: this prevents halting the CPU.

**VX\_POWER\_MODE\_AUTOHALT** (0x4)

Power management is enable: this allows halting the CPU.

**RETURNS**

OK, or **ERROR** if *mode* is incorrect or not supported by the processor.

**SEE ALSO**

**vxLib**, **vxPowerModeGet()**, **vxPowerDown()**

## **vxSSDisable()**

<b>NAME</b>	<b>vxSSDisable()</b> – disable the superscalar dispatch (MC68060)
<b>SYNOPSIS</b>	<code>void vxSSDisable (void)</code>
<b>DESCRIPTION</b>	This function resets the ESS bit of the Processor Configuration Register (PCR) to disable the superscalar dispatch.
<b>RETURNS</b>	N/A
<b>SEE ALSO</b>	<b>vxLib</b>

---

## **vxSSEnable()**

<b>NAME</b>	<b>vxSSEnable()</b> – enable the superscalar dispatch (MC68060)
<b>SYNOPSIS</b>	<code>void vxSSEnable (void)</code>
<b>DESCRIPTION</b>	This function sets the ESS bit of the Processor Configuration Register (PCR) to enable the superscalar dispatch.
<b>RETURNS</b>	N/A
<b>SEE ALSO</b>	<b>vxLib</b>

---

## vxTas()

<b>NAME</b>	<b>vxTas()</b> – C-callable atomic test-and-set primitive
<b>SYNOPSIS</b>	<pre>BOOL vxTas (     void * address          /* address to test and set */ )</pre>
<b>DESCRIPTION</b>	<p>This routine provides a C-callable interface to a test-and-set instruction. The test-and-set instruction is executed on the specified address. The architecture test-and-set instruction is:</p> <p>68K: <b>tas</b> x86: <b>lock bts</b> SH: <b>tas.b</b> ARM <b>swpb</b></p> <p>This routine is equivalent to <b>sysBusTas()</b> in <b>sysLib</b>.</p>
<b>NOTE MIPS</b>	Because VxWorks does not support the MIPS MMU, only <b>kseg0</b> and <b>kseg1</b> addresses are accepted; other addresses return <b>FALSE</b> .
<b>NOTE x86</b>	<b>BTS</b> “Bit Test and Set” instruction is executed with <b>LOCK</b> instruction prefix to lock the Bus during the execution. The bit position 0 is toggled.
<b>NOTE SH</b>	The SH version of <b>vxTas()</b> simply executes the <b>tas.b</b> instruction, and the test-and-set (atomic read-modify-write) operation may require an external bus locking mechanism on some hardware. In this case, wrap the <b>vxTas()</b> with a bus locking and unlocking code in the <b>sysBusTas()</b> .
<b>RETURNS</b>	<b>TRUE</b> if the value had not been set (but is now), or <b>FALSE</b> if the value was set already.
<b>SEE ALSO</b>	<b>vxLib</b> , <b>sysBusTas()</b>

## **vxTssGet()**

**NAME** vxTssGet() – get a content of the TASK register (x86)

**SYNOPSIS** `int vxTssGet (void)`

**DESCRIPTION** This routine gets a content of the TASK register

**RETURNS** a value of the TASK register

**SEE ALSO** vxLib

---

## **vxTssSet()**

**NAME** vxTssSet() – set a value to the TASK register (x86)

**SYNOPSIS** `void vxTssSet  
(  
    int value                   /* TASK register value */  
)`

**DESCRIPTION** This routine sets a value to the TASK register

**RETURNS** N/A

**SEE ALSO** vxLib

---

## wcstombs()

<b>NAME</b>	<b>wcstombs()</b> – convert a series of wide char’s to multibyte char’s (Unimplemented) (ANSI)
<b>SYNOPSIS</b>	<pre>size_t wcstombs (     char *          s,     const wchar_t * pwcs,     size_t         n )</pre>
<b>DESCRIPTION</b>	This multibyte character function is unimplemented in VxWorks.
<b>INCLUDE FILES</b>	stdlib.h
<b>RETURNS</b>	OK, or ERROR if the parameters are invalid.
<b>SEE ALSO</b>	ansiStdlib

---

## wctomb()

<b>NAME</b>	<b>wctomb()</b> – convert a wide character to a multibyte character (Unimplemented) (ANSI)
<b>SYNOPSIS</b>	<pre>int wctomb (     char * s,     wchar_t wchar )</pre>
<b>DESCRIPTION</b>	This multibyte character function is unimplemented in VxWorks.
<b>INCLUDE FILES</b>	stdlib.h
<b>RETURNS</b>	OK, or ERROR if the parameters are invalid.
<b>SEE ALSO</b>	ansiStdlib

## wdbSystemSuspend()

**NAME** wdbSystemSuspend() – suspend the system.

**SYNOPSIS** `STATUS wdbSystemSuspend (void)`

**DESCRIPTION** This routine transfers control from the run time system to the WDB agent running in external mode. In order to give back the control to the system it must be resumed by the the external WDB agent.

**EXAMPLE** The code below, called in a vxWorks application, suspends the system:

```
if (wdbSystemSuspend != OK)
    printf ("External mode is not supported by the WDB agent.\n");
```

From a host tool, we can detect that the system is suspended.

First, attach to the target server:

```
wtxtcl> wtxToolAttach EP960CX
EP960CX_ps@sevre
```

Then, you can get the agent mode:

```
wtxtcl> wtxAgentModeGet
AGENT_MODE_EXTERN
```

To get the status of the system context, execute:

```
wtxtcl> wtxContextStatusGet CONTEXT_SYSTEM 0
CONTEXT_SUSPENDED
```

In order to resume the system, simply execute:

```
wtxtcl> wtxContextResume CONTEXT_SYSTEM 0
0
```

You will see that the system is now running:

```
wtxtcl> wtxContextStatusGet CONTEXT_SYSTEM 0
CONTEXT_RUNNING
```

**RETURNS** OK upon successful completion, ERROR if external mode is not supported by the WDB agent.

**SEE ALSO** wdbLib

---

## wdbUserEvtLibInit()

<b>NAME</b>	<code>wdbUserEvtLibInit()</code> – include the WDB user event library
<b>SYNOPSIS</b>	<code>void wdbUserEvtLibInit (void)</code>
<b>DESCRIPTION</b>	This null routine is provided so that <code>wdbUserEvtLib</code> can be linked into the system. If <code>INCLUDE_WDB_USER_EVENT</code> is defined in <code>configAll.h</code> , <code>wdbUserEvtLibInit()</code> is called by the WDB config routine, <code>wdbConfig()</code> , in <code>usrWdb.c</code> .
<b>RETURNS</b>	N/A
<b>SEE ALSO</b>	<code>wdbUserEvtLib</code>

---

## wdbUserEvtPost()

<b>NAME</b>	<code>wdbUserEvtPost()</code> – post a user event string to host tools.
<b>SYNOPSIS</b>	<pre> STATUS wdbUserEvtPost (     char * event          /* event string to send */ ) </pre>
<b>DESCRIPTION</b>	This routine posts the string <i>event</i> to host tools that have registered for it. Host tools will receive a USER WTX event string. The maximum size of the event is <code>WDB_MAX_USER_EVT_SIZE</code> (defined in <code>\$WIND_BASE/target/h/wdb/wdbLib.h</code> ).

**EXAMPLE** The code below sends a WDB user event to host tools:

```

char * message = "Alarm: reactor overheating !!!";
if (wdbUserEvtPost (message) != OK)
    printf ("Can't send alarm message to host tools");

```

This event will be received by host tools that have registered for it. For example a WTX TCL based tool would do:

```

wtxtcl> wtxToolAttach EP960CX
EP960CX_ps@sevre
wtxtcl> wtxRegisterForEvent "USER.*"
0
wtxtcl> wtxEventGet

```

```
USER Alarm: reactor overheating !!!
```

Host tools can register for more specific user events:

```
wtxctl> wtxToolAttach EP960CX
EP960CX_ps@sevre
wtxctl> wtxRegisterForEvent "USER Alarm.*"
0
wtxctl> wtxEventGet
USER Alarm: reactor overheating !!!
```

In this piece of code, only the USER events beginning with “Alarm” will be received.

- RETURNS** OK upon successful completion, a WDB error code if unable to send the event to the host or ERROR if the size of the event is greater than WDB\_MAX\_USER\_EVT\_SIZE.
- SEE ALSO** wdbUserEvtLib

---

## wdCancel()

- NAME** wdCancel() – cancel a currently counting watchdog
- SYNOPSIS**
- ```
STATUS wdCancel
(
    WDOG_ID wdId          /* ID of watchdog to cancel */
)
```
- DESCRIPTION** This routine cancels a currently running watchdog timer by zeroing its delay count. Watchdog timers may be canceled from interrupt level.
- RETURNS** OK, or ERROR if the watchdog timer cannot be canceled.
- SEE ALSO** wdLib, wdStart()

---

## wdCreate()

**NAME** `wdCreate()` – create a watchdog timer

**SYNOPSIS** `WDOG_ID wdCreate (void)`

**DESCRIPTION** This routine creates a watchdog timer by allocating a WDOG structure in memory.

**RETURNS** The ID for the watchdog created, or `NULL` if memory is insufficient.

**SEE ALSO** `wdLib`, `wdDelete()`

---

## wdDelete()

**NAME** `wdDelete()` – delete a watchdog timer

**SYNOPSIS** `STATUS wdDelete`  
`(`  
`WDOG_ID wdId                  /* ID of watchdog to delete */`  
`)`

**DESCRIPTION** This routine de-allocates a watchdog timer. The watchdog will be removed from the timer queue if it has been started. This routine complements `wdCreate()`.

**RETURNS** `OK`, or `ERROR` if the watchdog timer cannot be de-allocated.

**SEE ALSO** `wdLib`, `wdCreate()`

## wdShow()

- NAME** `wdShow()` – show information about a watchdog
- SYNOPSIS**
- ```
STATUS wdShow  
(  
    WDOG_ID wdId          /* watchdog to display */  
)
```
- DESCRIPTION** This routine displays the state of a watchdog.
- EXAMPLE** A summary of the state of a watchdog is displayed as follows:
- ```
-> wdShow myWdId  
Watchdog Id      : 0x3dd46c  
State            : OUT_OF_Q  
Ticks Remaining  : 0  
Routine          : 0  
Parameter        : 0
```
- RETURNS** OK or ERROR.
- SEE ALSO** `wdShow`, *VxWorks Programmer's Guide: Target Shell*, `windsh`, *Tornado User's Guide: Shell*
- 

## wdShowInit()

- NAME** `wdShowInit()` – initialize the watchdog show facility
- SYNOPSIS**
- ```
void wdShowInit (void)
```
- DESCRIPTION** This routine links the watchdog show facility into the VxWorks system. It is called automatically when the watchdog show facility is configured into VxWorks using either of the following methods:
- If you use the configuration header files, define `INCLUDE_SHOW_ROUTINES` in `config.h`.
  - If you use the Tornado project facility, select `INCLUDE_WATCHDOGS_SHOW`.
- RETURNS** N/A
- SEE ALSO** `wdShow`

---

## wdStart()

**NAME** wdStart() – start a watchdog timer

**SYNOPSIS**

```
STATUS wdStart
(
    WDOG_ID wdId,           /* watchdog ID */
    int delay,             /* delay count, in ticks */
    FUNCPTR pRoutine,     /* routine to call on time-out */
    int parameter         /* parameter with which to call routine */
)
```

**DESCRIPTION** This routine adds a watchdog timer to the system tick queue. The specified watchdog routine will be called from interrupt level after the specified number of ticks has elapsed. Watchdog timers may be started from interrupt level.

To replace either the timeout *delay* or the routine to be executed, call **wdStart()** again with the same *wdId*; only the most recent **wdStart()** on a given watchdog ID has any effect. (If your application requires multiple watchdog routines, use **wdCreate()** to generate separate a watchdog ID for each.) To cancel a watchdog timer before the specified tick count is reached, call **wdCancel()**.

Watchdog timers execute only once, but some applications require periodically executing timers. To achieve this effect, the timer routine itself must call **wdStart()** to restart the timer on each invocation.

---

**WARNING:** The watchdog routine runs in the context of the system-clock ISR; thus, it is subject to all ISR restrictions.

---

**RETURNS** OK, or **ERROR** if the watchdog timer cannot be started.

**SEE ALSO** wdLib, wdCancel()

## whoami()

<b>NAME</b>	<b>whoami()</b> – display the current remote identity
<b>SYNOPSIS</b>	<code>void whoami (void)</code>
<b>DESCRIPTION</b>	This routine displays the user name currently used for remote machine access. The user name is set with <b>iam()</b> or <b>remCurIdSet()</b> .
<b>RETURNS</b>	N/A
<b>SEE ALSO</b>	<b>remLib</b> , <b>iam()</b> , <b>remCurIdGet()</b> , <b>remCurIdSet()</b>

---

## write()

<b>NAME</b>	<b>write()</b> – write bytes to a file
<b>SYNOPSIS</b>	<pre>int write (     int    fd,                /* file descriptor on which to write */     char * buffer,           /* buffer containing bytes to be written */     size_t nbytes           /* number of bytes to write */ )</pre>
<b>DESCRIPTION</b>	This routine writes <i>nbytes</i> bytes from <i>buffer</i> to a specified file descriptor <i>fd</i> . It calls the device driver to do the work.
<b>RETURNS</b>	The number of bytes written (if not equal to <i>nbytes</i> , an error has occurred), or <b>ERROR</b> if the file descriptor does not exist, the driver does not have a write routine, or the driver returns <b>ERROR</b> . If the driver does not have a write routine, <b>errno</b> is set to <b>ENOTSUP</b> .
<b>SEE ALSO</b>	<b>ioLib</b>

---

## wvEvent()

**NAME** wvEvent() – log a user-defined event (WindView)

**SYNOPSIS**

```
STATUS wvEvent
(
    event_t usrEventId,      /* event */
    char *  buffer,         /* buffer */
    size_t  bufSize        /* buffer size */
)
```

**DESCRIPTION** This routine logs a user event. Event logging must have been started with **wvEvtLogEnable()** or from the WindView GUI to use this routine. The *usrEventId* should be in the range 0-25535. A buffer of data can be associated with the event; *buffer* is a pointer to the start of the data block, and *bufSize* is its length in bytes. The size of the event buffer configured with **wvInstInit()** should be adjusted when logging large user events.

**RETURNS** OK, or ERROR if the event can not be logged.

**SEE ALSO** wvLib, dbgLib, e()

---

## wvEventInst()

**NAME** wvEventInst() – instrument VxWorks Events (WindView)

**SYNOPSIS**

```
STATUS wvEventInst
(
    int mode                /* instrumentation mode */
)
```

**DESCRIPTION** This routine instruments VxWorks Event activity.

If *mode* is **INSTRUMENT\_ON**, instrumentation for events is turned on; if it is any other value (including **INSTRUMENT\_OFF**), instrumentation for VxWorks Events is turned off.

This routine has effect only if **INCLUDE\_WINDVIEW** is defined in **configAll.h** and event logging has been enabled for system objects.

**RETURNS** OK or ERROR.

**SEE ALSO** wvLib

## wvEvtBufferGet()

**NAME** `wvEvtBufferGet()` – return the ID of the WindView event buffer (WindView)

**SYNOPSIS** `BUFFER_ID wvEvtBufferGet (void)`

**RETURNS** The event buffer ID if one exists, otherwise NULL.

**SEE ALSO** `wvLib`

---

## wvEvtClassClear()

**NAME** `wvEvtClassClear()` – clear a class of events from those being logged (WindView)

**SYNOPSIS**

```
void wvEvtClassClear
(
    UINT32 classDescription /* description of evt classes to clear */
)
```

**DESCRIPTION** This routine clears the class or classes described by *classDescription* from the set of classes currently being logged.

**RETURNS** N/A

**SEE ALSO** `wvLib`

---

## wvEvtClassClearAll()

**NAME** `wvEvtClassClearAll()` – clear all classes of events from those logged (WindView)

**SYNOPSIS** `void wvEvtClassClearAll (void)`

**DESCRIPTION** This routine clears all classes of events so that no classes are logged if event logging is started.

**RETURNS** N/A

**SEE ALSO** `wvLib`

---

## wvEvtClassGet()

**NAME** wvEvtClassGet() – get the current set of classes being logged (WindView)

**SYNOPSIS** `UINT32 wvEvtClassGet (void)`

**DESCRIPTION** This routine returns the set of classes currently being logged.

**RETURNS** The class description.

**SEE ALSO** wvLib

---

## wvEvtClassSet()

**NAME** wvEvtClassSet() – set the class of events to log (WindView)

**SYNOPSIS**

```
void wvEvtClassSet
(
    UINT32 classDescription /* description of evt classes to set */
)
```

**DESCRIPTION** This routine sets the class of events which are logged when event logging is started. *classDescription* can take the following values:

```
WV_CLASS_1    /* Events causing context switches */
WV_CLASS_2    /* Events causing task-state transitions */
WV_CLASS_3    /* Events from object and system libraries */
```

See **wvLib** for more information about these classes, particularly Class 3.

**RETURNS** N/A

**SEE ALSO** wvLib, wvObjInst(), wvObjInstModeSet(), wvSigInst(), wvEventInst()

## wvEvtLogInit()

<b>NAME</b>	<b>wvEvtLogInit()</b> – initialize an event log (WindView)
<b>SYNOPSIS</b>	<pre>void wvEvtLogInit (     BUFFER_ID evtBufId      /* event-buffer id */ )</pre>
<b>DESCRIPTION</b>	This routine initializes event logging by associating a particular event buffer with the logging functions. It must be called before event logging is turned on.
<b>RETURNS</b>	N/A
<b>SEE ALSO</b>	<b>wvLib</b>

---

## wvEvtLogStart()

<b>NAME</b>	<b>wvEvtLogStart()</b> – start logging events to the buffer (WindView)
<b>SYNOPSIS</b>	<pre>void wvEvtLogStart (void)</pre>
<b>DESCRIPTION</b>	This routine starts event logging. It also resets the timestamp mechanism so that it can be called more than once without stopping event logging.
<b>RETURNS</b>	N/A
<b>SEE ALSO</b>	<b>wvLib</b>

---

## wvEvtLogStop()

<b>NAME</b>	<b>wvEvtLogStop()</b> – stop logging events to the buffer (WindView)
<b>SYNOPSIS</b>	<b>void wvEvtLogStop (void)</b>
<b>DESCRIPTION</b>	This routine turns off all event logging, including event-logging of objects and signals specifically requested by the user. In addition, it disables the timestamp facility.
<b>RETURNS</b>	N/A
<b>SEE ALSO</b>	<b>wvLib</b>

---

## wvLibInit()

<b>NAME</b>	<b>wvLibInit()</b> – initialize <b>wvLib</b> - first step (WindView)
<b>SYNOPSIS</b>	<b>void wvLibInit (void)</b>
<b>DESCRIPTION</b>	This routine starts initializing <b>wvLib</b> . Its actions should be performed before object creation, so it is called from <b>usrKernelInit()</b> in <b>usrKernel.c</b> .
<b>RETURNS</b>	N/A
<b>SEE ALSO</b>	<b>wvLib</b>

---

## wvLibInit2()

<b>NAME</b>	<b>wvLibInit2()</b> – initialize <b>wvLib</b> - final step (WindView)
<b>SYNOPSIS</b>	<b>void wvLibInit2 (void)</b>
<b>DESCRIPTION</b>	This routine is called after <b>wvLibInit()</b> to complete the initialization of <b>wvLib</b> . It should be called before starting any event logging.
<b>RETURNS</b>	N/A
<b>SEE ALSO</b>	<b>wvLib</b>

---

## wvLogHeaderCreate()

**NAME** wvLogHeaderCreate() – create the event-log header (WindView)

**SYNOPSIS**

```
WV_LOG_HEADER_ID wvLogHeaderCreate
(
    PART_ID memPart          /* partition where header should be stored */
)
```

**DESCRIPTION** This routine creates the header of `EVENT_CONFIG`, `EVENT_BUFFER`, and `EVENT_BEGIN` events that is required at the beginning of every event log. These events are stored in a packed array allocated from the specified memory partition. In addition to this separate header, this routine also logs all tasks active in the system to the event buffer for uploading along with the other events.

This routine should be called after `wvEvtLogInit()` is called. If uploading events continuously to the host, this routine should be called after the upload task is started. This ensures that the upload task is included in the snapshot of active tasks. If upload will occur after event logging has stopped (deferred upload), this routine can be called any time before event logging is turned on.

**RETURNS** A valid `WV_LOG_HEADER_ID`, or `NULL` if memory can not be allocated.

**SEE ALSO** wvLib

---

## wvLogHeaderUpload()

**NAME** wvLogHeaderUpload() – transfer the log header to the host (WindView)

**SYNOPSIS**

```
STATUS wvLogHeaderUpload
(
    WV_LOG_HEADER_ID pHeader, /* pointer to the header */
    UPLOAD_ID pathId /* path by which to upload to host */
)
```

**DESCRIPTION** This functions transfers the log header events (`EVENT_BEGIN`, `EVENT_CONFIG`, `EVENT_BUFFER`) to the host. These events were saved to a local buffer with the call to `wvLogHeaderCreate()`. This routine should be called before any events or task names are uploaded to the host. The events in the header buffer must be the first things the parser sees.

If continuously uploading events, it is best to start the uploader, and then call this routine. If deferring upload until after event logging is stopped, this should be called before the uploader is started.

**RETURNS** OK, or **ERROR** if there is trouble with the upload path.

**SEE ALSO** wvLib

---

## wvNetAddressFilterClear()

**NAME** wvNetAddressFilterClear() – remove the address filter for events

**SYNOPSIS**

```
void wvNetAddressFilterClear
(
    int type,                /* 0 for source, 1 for destination */
    int direction            /* 0 for input, 1 for output */
)
```

**DESCRIPTION** This routine removes any active address filter test indicated by the *type* and *direction* parameters used to enable it. Affected events will be reported unconditionally.

**RETURNS** N/A

**SEE ALSO** wvNetLib

---

## wvNetAddressFilterSet()

**NAME** wvNetAddressFilterSet() – specify an address filter for events

**SYNOPSIS**

```
STATUS wvNetAddressFilterSet
(
    char * pAddress,        /* target address for event comparisons */
    char * pMask,          /* mask value applied to data fields */
    int type,              /* 0 for source, 1 for destination */
    int direction          /* 0 for input, 1 for output */
)
```



<b>DESCRIPTION</b>	This routine activates an additional test that disables certain events that do not match the specified IP address. The <i>pAddress</i> parameter provides the test value in dotted-decimal format. The <i>type</i> parameter indicates whether that address is compared against the source or destination values, and the <i>direction</i> value identifies whether the <i>type</i> is interpreted from the perspective of incoming or outgoing traffic. The <i>pMask</i> parameter provides a network mask to support testing for a group of events.
<b>RETURNS</b>	OK if filter set, or <b>ERROR</b> otherwise.
<b>ERRNO</b>	N/A
<b>SEE ALSO</b>	<b>wvNetLib</b>

---

## wvNetDisable()

<b>NAME</b>	<b>wvNetDisable()</b> – end reporting of network events to WindView
<b>SYNOPSIS</b>	<b>void wvNetDisable (void)</b>
<b>DESCRIPTION</b>	This routine stops WindView event reporting for all network components.
<b>RETURNS</b>	N/A
<b>ERRNO</b>	N/A
<b>SEE ALSO</b>	<b>wvNetLib</b>

---

## wvNetEnable()

<b>NAME</b>	<b>wvNetEnable()</b> – begin reporting network events to WindView
<b>SYNOPSIS</b>	<pre>void wvNetEnable (     int priority                /* minimum priority, or 0 for default of */                                 /* WV_NET_VERBOSE */ )</pre>

<b>DESCRIPTION</b>	<p>This routine activates WindView event reporting for network components, after disabling all events with a priority less than <i>level</i>. The default value (or a <i>level</i> of <b>WV_NET_VERBOSE</b>) will not disable any additional events. The available priority values are:</p> <ul style="list-style-type: none"> <li><b>WV_NET_EMERGENCY</b> (1)</li> <li><b>WV_NET_ALERT</b> (2)</li> <li><b>WV_NET_CRITICAL</b> (3)</li> <li><b>WV_NET_ERROR</b> (4)</li> <li><b>WV_NET_WARNING</b> (5)</li> <li><b>WV_NET_NOTICE</b> (6)</li> <li><b>WV_NET_INFO</b> (7)</li> <li><b>WV_NET_VERBOSE</b> (8)</li> </ul> <p>If an event is not explicitly disabled by the priority level, it uses the current event selection map and class settings. The initial values enable all events of both classes.</p>
<b>RETURNS</b>	N/A
<b>ERRNO</b>	N/A
<b>SEE ALSO</b>	<b>wvNetLib</b>

---

## wvNetEventDisable()

<b>NAME</b>	<b>wvNetEventDisable()</b> – deactivate specific network events
<b>SYNOPSIS</b>	<pre> <b>STATUS</b> wvNetEventDisable (     int priority,           /* priority level of event */     int offset             /* identifier within priority level */ ) </pre>
<b>DESCRIPTION</b>	<p>This routine prevents reporting of a single event within the priority equal to <i>level</i>. The activation is overridden if the setting for the entire priority level changes. The available priority values are:</p> <ul style="list-style-type: none"> <li><b>WV_NET_EMERGENCY</b> (1)</li> <li><b>WV_NET_ALERT</b> (2)</li> <li><b>WV_NET_CRITICAL</b> (3)</li> <li><b>WV_NET_ERROR</b> (4)</li> <li><b>WV_NET_WARNING</b> (5)</li> <li><b>WV_NET_NOTICE</b> (6)</li> </ul>



WV\_NET\_INFO (7)  
WV\_NET\_VERBOSE (8)

Offset values for individual events are listed in the documentation.

**RETURNS** OK, or ERROR for unknown event.

**ERRNO** N/A

**SEE ALSO** wvNetLib

---

## wvNetEventEnable()

**NAME** wvNetEventEnable() – activate specific network events

**SYNOPSIS**

```
STATUS wvNetEventEnable  
(  
    int priority,           /* priority level of event */  
    int offset             /* identifier within priority level */  
)
```

**DESCRIPTION** This routine allows reporting of a single event within the priority equal to *level*. The activation is overridden if the setting for the entire priority level changes. The available priority values are:

WV\_NET\_EMERGENCY (1)  
WV\_NET\_ALERT (2)  
WV\_NET\_CRITICAL (3)  
WV\_NET\_ERROR (4)  
WV\_NET\_WARNING (5)  
WV\_NET\_NOTICE (6)  
WV\_NET\_INFO (7)  
WV\_NET\_VERBOSE (8)

Offset values for individual events are listed in the documentation.

**RETURNS** OK, or ERROR for unknown event.

**ERRNO** N/A

**SEE ALSO** wvNetLib

---

## wvNetLevelAdd()

**NAME** wvNetLevelAdd() – enable network events with specific priority level

**SYNOPSIS**

```
STATUS wvNetLevelAdd
(
    int priority          /* priority level to enable */
)
```

**DESCRIPTION** This routine changes the event selection map to allow reporting of any events with priority equal to *level*. It will override current event selections for the given priority, but has no effect on settings for events with higher or lower priorities. The available priority values are:

```
WV_NET_EMERGENCY (1)
WV_NET_ALERT (2)
WV_NET_CRITICAL (3)
WV_NET_ERROR (4)
WV_NET_WARNING (5)
WV_NET_NOTICE (6)
WV_NET_INFO (7)
WV_NET_VERBOSE (8)
```

Events are only reported based on the current WindView class setting. The initial (default) setting includes networking events from both classes.

**RETURNS** OK, or ERROR for unknown event level.

**ERRNO** N/A

**SEE ALSO** wvNetLib

---

## wvNetLevelRemove()

**NAME** wvNetLevelRemove() – disable network events with specific priority level

**SYNOPSIS**

```
STATUS wvNetLevelRemove
(
    int priority          /* priority level to disable */
)
```

**DESCRIPTION** This routine changes the event selection map to prevent reporting of any events with priority equal to *level*. It will override the current event selection for the given priority, but has no effect on settings for events with higher or lower priorities. The available priority values are:

- WV\_NET\_EMERGENCY (1)
- WV\_NET\_ALERT (2)
- WV\_NET\_CRITICAL (3)
- WV\_NET\_ERROR (4)
- WV\_NET\_WARNING (5)
- WV\_NET\_NOTICE (6)
- WV\_NET\_INFO (7)
- WV\_NET\_VERBOSE (8)

Events are only reported based on the current WindView class setting. The initial (default) setting includes networking events from both classes.

**RETURNS** OK, or ERROR for unknown event level.

**ERRNO** N/A

**SEE ALSO** wvNetLib

---

## wvNetPortFilterClear()

**NAME** wvNetPortFilterClear() – remove the port number filter for events

**SYNOPSIS**

```
void wvNetPortFilterClear
(
    int type,                /* 0 for source, 1 for destination */
    int direction           /* 0 for input, 1 for output */
)
```

**DESCRIPTION** This routine removes any active port filter test indicated by the *type* and *direction* parameters used to enable it. Affected events will be reported unconditionally.

**RETURNS** N/A

**ERRNO** N/A

**SEE ALSO** wvNetLib

---

## wvNetPortFilterSet()

**NAME** wvNetPortFilterSet() – specify an address filter for events

**SYNOPSIS**

```
STATUS wvNetPortFilterSet
(
    int port,                /* target port for event comparisons */
    int type,                /* 0 for source, 1 for destination */
    int direction            /* 0 for input, 1 for output */
)
```

**DESCRIPTION** This routine activates an additional filter, which disables certain events that do not match the specified port value. The *port* parameter provides the test value and the *type* parameter indicates whether that value is compared against the source or destination fields. The *direction* setting identifies whether the *type* is interpreted from the perspective of incoming or outgoing traffic.

**RETURNS** OK if filter set, or ERROR otherwise.

**ERRNO** N/A

**SEE ALSO** wvNetLib

---

## wvObjInst()

**NAME** wvObjInst() – instrument objects (WindView)

**SYNOPSIS**

```
STATUS wvObjInst
(
    int objType,            /* object type */
    void * objId,          /* object ID or NULL for all objects */
    int mode                /* instrumentation mode */
)
```

**DESCRIPTION** This routine instruments a specified object or set of objects and has effect when system objects have been enabled for event logging.

*objType* can be set to one of the following: OBJ\_TASK (tasks), OBJ\_SEM (semaphores), OBJ\_MSG (message queues), or OBJ\_WD (watchdogs). *objId* specifies the identifier of the

particular object to be instrumented. If *objId* is `NULL`, then all objects of *objType* have instrumentation turned on or off depending on the value of *mode*.

If *mode* is `INSTRUMENT_ON`, instrumentation is turned on; if it is any other value (including `INSTRUMENT_OFF`) then instrumentation is turned off for *objId*.

Call `wvObjInstModeSet( )` with `INSTRUMENT_ON` if you want to enable instrumentation for all objects created after a certain place in your code. Use `wvSigInst( )` if you want to enable instrumentation for all signal activity.

This routine has effect only if `INCLUDE_WINDVIEW` is defined in `configAll.h`.

**RETURNS** OK or `ERROR`.

**SEE ALSO** `wvLib`, `wvSigInst( )`, `wvEventInst( )`, `wvObjInstModeSet( )`

---

## wvObjInstModeSet( )

**NAME** `wvObjInstModeSet( )` – set object instrumentation on/off (WindView)

**SYNOPSIS**

```
STATUS wvObjInstModeSet
(
    int mode                /* object instrumentation on/off */
)
```

**DESCRIPTION** This routine causes objects to be created either instrumented or not depending on the value of *mode*, which can be `INSTRUMENT_ON` or `INSTRUMENT_OFF`. All objects created after `wvObjInstModeSet( )` is called with `INSTRUMENT_ON` and before it is called with `INSTRUMENT_OFF` are created as instrumented objects.

Use `wvObjInst( )` if you want to enable instrumentation for a specific object or set of objects. Use `wvSigInst( )` if you want to enable instrumentation for all signal activity, and `wvEventInst( )` to enable instrumentation for VxWorks Event activity.

This routine has effect only if `INCLUDE_WINDVIEW` is defined in `configAll.h`.

**RETURNS** The previous value of *mode* or `ERROR`.

**SEE ALSO** `wvLib`, `wvObjInst( )`, `wvSigInst( )`, `wvEventInst( )`

---

## wvRBuffMgrPrioritySet()

<b>NAME</b>	<b>wvRBuffMgrPrioritySet()</b> – set the priority of the WindView rBuff manager (WindView)
<b>SYNOPSIS</b>	<pre><b>STATUS wvRBuffMgrPrioritySet</b>   (     <b>int priority</b>                <i>/* new priority */</i>   )</pre>
<b>DESCRIPTION</b>	This routine changes the priority of the <b>tWvRBuffMgr</b> task to the value of <i>priority</i> . Priorities range from 0, the highest priority, to 255, the lowest priority. If the task is not yet running, this priority is used when it is spawned.
<b>RETURNS</b>	OK, or ERROR if the priority can not be set.
<b>SEE ALSO</b>	<b>rBuffLib</b> , <b>taskPrioritySet()</b> , <i>VxWorks Programmer's Guide: Basic OS</i>

---

## wvSigInst()

<b>NAME</b>	<b>wvSigInst()</b> – instrument signals (WindView)
<b>SYNOPSIS</b>	<pre><b>STATUS wvSigInst</b>   (     <b>int mode</b>                    <i>/* instrumentation mode */</i>   )</pre>
<b>DESCRIPTION</b>	<p>This routine instruments all signal activity.</p> <p>If <i>mode</i> is <b>INSTRUMENT_ON</b>, instrumentation for signals is turned on; if it is any other value (including <b>INSTRUMENT_OFF</b>), instrumentation for signals is turned off.</p> <p>This routine has effect only if <b>INCLUDE_WINDVIEW</b> is defined in <b>configAll.h</b> and event logging has been enabled for system objects.</p>
<b>RETURNS</b>	OK or ERROR.
<b>SEE ALSO</b>	<b>wvLib</b>

---

## wvTaskNamesPreserve()

**NAME** `wvTaskNamesPreserve()` – preserve an extra copy of task name events (WindView)

**SYNOPSIS**

```
TASKBUF_ID wvTaskNamesPreserve
(
    PART_ID memPart,          /* memory where preserved names are stored */
    int     size              /* must be a power of 2 */
)
```

**DESCRIPTION** This routine initializes the data structures and instrumentation necessary to allow WindView to store an extra copy of each `EVENT_TASKNAME` event, which is necessary for post-mortem analysis. This routine should be called after `wvEvtLogInit()` has been called, and before event logging is started.

If this routine is called before event logging is started, all `EVENT_TASKNAME` events that are produced by VxWorks are logged into the standard event buffer, and a copy of each is logged automatically to the task name buffer created by this routine. All tasks running when this routine is called are also added to the buffer. The events in this buffer can be uploaded after the other events have been uploaded, to provide the task names for any events in the log which no longer have a corresponding task name event due to wrapping of data in the buffers. Because there may be two copies of some of the task name events after the buffer data wraps around, the resultant log may have two task name events for the same task. This is not a problem for the parser.

Occasionally the task ID of a task is reused, and in this case, only the last instance of the task name event with a particular task ID is maintained.

The buffer size must be a power of two.

This routine sets the event class `WV_CLASS_TASKNAMES_PRESERVE`, which can be turned off by calling `wvEvtClassClear()` or `wvEvtClassSet()`.

**RETURNS** A valid `TASKBUF_ID` to be used for later uploading, or `NULL` if not enough memory exists to create the task buffer.

**SEE ALSO** `wvLib`

---

## wvTaskNamesUpload()

**NAME** wvTaskNamesUpload() – upload preserved task name events (WindView)

**SYNOPSIS**

```
STATUS wvTaskNamesUpload
(
    TASKBUF_ID taskBufId,    /* taskname event buffer to upload */
    UPLOAD_ID pathId        /* upload path id */
)
```

**DESCRIPTION** This routine uploads task name events, saved after calling `wvTaskNamesPreserve()`, to the host by the specified upload path. There is no particular order to the events uploaded. All the events contained in the buffer are uploaded in one pass. After all have been uploaded, the buffer used to store the events is destroyed.

**RETURNS** OK, or ERROR if the upload path or task name buffer is invalid.

**SEE ALSO** wvLib

---

## wvTmrRegister()

**NAME** wvTmrRegister() – register a timestamp timer (WindView)

**SYNOPSIS**

```
void wvTmrRegister
(
    UINTEFUNCPTR wvTmrRtn,    /* timestamp routine */
    UINTEFUNCPTR wvTmrLockRtn, /* locked timestamp routine */
    FUNCPTR      wvTmrEnable, /* enable timer routine */
    FUNCPTR      wvTmrDisable, /* disable timer routine */
    FUNCPTR      wvTmrConnect, /* connect to timer routine */
    UINTEFUNCPTR wvTmrPeriod, /* period of timer routine */
    UINTEFUNCPTR wvTmrFreq    /* frequency of timer routine */
)
```

**DESCRIPTION** This routine registers a timestamp routine for each of the following:

*wvTmrRtn*

a timestamp routine, which returns a timestamp when called (must be called with interrupts locked).

**wvUploadStart( )***wvTmrLockRtn*

a timestamp routine, which returns a timestamp when called (locks interrupts).

*wvTmrEnable*

an enable-timer routine, which enables the timestamp timer.

*wvTmrDisable*

a disable-timer routine, which disables the timestamp timer.

*wvTmrConnect*

a connect-to-timer routine, which connects a handler to be run when the timer rolls over; this routine should return **NULL** if the system clock tick is to be used.

*wvTmrPeriod*

a period-of-timer routine, which returns the period of the timer.

*wvTmrFreq*

a frequency-of-timer routine, which returns the frequency of the timer.

If any of these routines is set to **NULL**, the behavior of instrumented code is undefined.

**RETURNS**

N/A

**SEE ALSO**

**wvTmrLib**

---

## wvUploadStart( )

**NAME**

**wvUploadStart( )** – start upload of events to the host (WindView)

**SYNOPSIS**

```
WV_UPLOADTASK_ID wvUploadStart
(
    BUFFER_ID bufId,           /* event data buffer ID */
    UPLOAD_ID pathId,         /* upload path to host */
    BOOL      uploadContinuously /* upload continuously if true */
)
```

**DESCRIPTION**

This routine starts uploading events from the event buffer to the host. Events can be uploaded either continuously or in one pass until the buffer is emptied. If *uploadContinuously* is set to **TRUE**, the task uploading events pends until more data arrives in the buffer. If **FALSE**, the buffer is flushed without waiting, but this routine returns immediately with an ID that can be used to kill the upload task. Upload is done by spawning the task **tWVUpload**. The buffer to upload is identified by *bufId*, and the upload path to use is identified by *pathId*.

This routine blocks if no event data is in the buffer, so it should be called before event logging is started to ensure the buffer does not overflow.

**RETURNS** A valid **WV\_UPLOADTASK\_ID** if started for continuous upload, a non-NULL value if started for one-pass upload, and **NULL** if the task can not be spawned or memory for the descriptor can not be allocated.

**SEE ALSO** **wvLib**

---

## wvUploadStop()

**NAME** **wvUploadStop()** – stop upload of events to host (WindView)

**SYNOPSIS**

```
STATUS wvUploadStop  
(  
    WV_UPLOADTASK_ID upTaskId  
)
```

**DESCRIPTION** This routine stops continuous upload of events to the host. It does this by making a request to the upload task to terminate after it has emptied the buffer. For this reason it is important to make sure data is no longer being logged to the buffer before calling this routine.

This task blocks until the buffer is emptied, and then frees memory associated with *upTaskId*.

**RETURNS** **OK** if the upload task terminates successfully, or **ERROR** either if *upTaskId* is invalid or if the upload task terminates with an **ERROR**.

**SEE ALSO** **wvLib**

## wvUploadTaskConfig()

<b>NAME</b>	<b>wvUploadTaskConfig()</b> – set priority and stack size of <b>tWVUpload</b> task (WindView)
<b>SYNOPSIS</b>	<pre>void wvUploadTaskConfig (     int stackSize,          /* the new stack size for tWVUpload */     int priority           /* the new priority for tWVUpload */ )</pre>
<b>DESCRIPTION</b>	This routine sets the stack size and priority of future instances of the event-data upload task, created by calling <b>wvUploadStart()</b> . The default stack size for this task is 5000 bytes, and the default priority is 150.
<b>RETURNS</b>	N/A
<b>SEE ALSO</b>	<b>wvLib</b>

---

## xattrib()

**NAME** xattrib() – modify MS-DOS file attributes of many files

**SYNOPSIS**

```
STATUS xattrib
(
    const char * source,      /* file or directory name on which to */
                             /* change flags */
    const char * attr        /* flag settings to change */
)
```

**DESCRIPTION** This function is essentially the same as `attrib()`, but it accepts wildcards in *fileName*, and traverses subdirectories in order to modify attributes of entire file hierarchies.

The *attr* argument string may contain must start with either “+” or “-”, meaning the attribute flags which will follow should be either set or cleared. After “+” or “-” any of these four letter will signify their respective attribute flags - “A”, “S”, “H” and “R”.

**EXAMPLE**

```
-> xattrib( "/sd0/sysfiles", "+RS") /* write protect "sysfiles" */
-> xattrib( "/sd0/logfiles", "-R") /* unprotect logfiles before deletion */
-> xdelete( "/sd0/logfiles")
```

---

**WARNING:** This function may call itself in accordance with the depth of the source directory, and occupies approximately 520 bytes per stack frame, meaning that to accommodate the maximum depth of subdirectories which is 20, at least 10 Kbytes of stack space should be available to avoid stack overflow.

---

**RETURNS** OK, or ERROR if the file can not be opened.

**SEE ALSO** `usrFsLib`

## xcopy()

**NAME** xcopy() – copy a hierarchy of files with wildcards

**SYNOPSIS**

```
STATUS xcopy
(
    const char * source,      /* source directory or wildcard name */
    const char * dest        /* destination directory */
)
```

**DESCRIPTION** *source* is a string containing a name of a directory, or a wildcard or both which will cause this function to make a recursive copy of all files residing in that directory and matching the wildcard pattern into the *dest* directory, preserving the file names and subdirectories.

---

**WARNING:** This function may call itself in accordance with the depth of the source directory, and occupies approximately 800 bytes per stack frame, meaning that to accommodate the maximum depth of subdirectories which is 20, at least 16 Kbytes of stack space should be available to avoid stack overflow.

---

**RETURNS** OK or ERROR if any operation has failed.

**SEE ALSO** usrFsLib, tarLib, checkStack(), cp()

---

## xdelete()

**NAME** xdelete() – delete a hierarchy of files with wildcards

**SYNOPSIS**

```
STATUS xdelete
(
    const char * source      /* source directory or wildcard name */
)
```

**DESCRIPTION** *source* is a string containing a name of a directory, or a wildcard or both which will cause this function to recursively remove all files and subdirectories residing in that directory and matching the wildcard pattern. When a directory is encountered, all its contents are removed, and then the directory itself is deleted.

---

**WARNING:** This function may call itself in accordance with the depth of the source directory, and occupies approximately 520 bytes per stack frame, meaning that to accommodate the maximum depth of subdirectories which is 20, at least 10 Kbytes of stack space should be available to avoid stack overflow.

---

**RETURNS** OK or **ERROR** if any operation has failed.

**SEE ALSO** **usrFsLib**, **checkStack()**, **cp()**, **copy()**, **xcopy()**, **tarLib**

## **zbufCreate()**

**NAME**            **zbufCreate()** – create an empty zbuf

**SYNOPSIS**        **ZBUF\_ID zbufCreate (void)**

**DESCRIPTION**    This routine creates a zbuf, which remains empty (that is, it contains no data) until segments are added by the zbuf insertion routines. Operations performed on zbufs require a zbuf ID, which is returned by this routine.

**VXWORKS AE PROTECTION DOMAINS**

Under VxWorks AE, you can call this function from within the kernel protection domain only. In addition, the returned ID is valid within the kernel protection domain only. This restriction does not apply under non-AE versions of VxWorks.

**RETURNS**        A zbuf ID, or NULL if a zbuf cannot be created.

**SEE ALSO**        **zbufLib, zbufDelete()**

---

## **zbufCut()**

**NAME**            **zbufCut()** – delete bytes from a zbuf

**SYNOPSIS**        **ZBUF\_SEG zbufCut**  
                  (  
                  **ZBUF\_ID zbufId,**            **/\* zbuf from which bytes are cut \*/**  
                  **ZBUF\_SEG zbufSeg,**        **/\* zbuf segment base for offset \*/**  
                  **int offset,**            **/\* relative byte offset \*/**  
                  **int len**             **/\* number of bytes to cut \*/**  
                  )

**DESCRIPTION**    This routine deletes *len* bytes from *zbufId* starting at the specified byte location.

The starting location of deletion is specified by *zbufSeg* and *offset*. See the **zbufLib** manual page for more information on specifying a byte location within a zbuf. In particular, the first byte deleted is the exact byte specified by *zbufSeg* and *offset*.

The number of bytes to delete is given by *len*. If this parameter is negative, or is larger than the number of bytes in the zbuf after the specified byte location, the rest of the zbuf is deleted. The bytes deleted may span more than one segment.

If all the bytes in any one segment are deleted, then the segment is deleted, and the data buffer that it referenced will be freed if no other zbuf segments reference it. No segment may survive with zero bytes referenced.

Deleting bytes out of the middle of a segment splits the segment into two. The first segment contains the portion of the data buffer before the deleted bytes, while the other segment contains the end portion that remains after deleting *len* bytes.

This routine returns the zbuf segment ID of the segment just after the deleted bytes. In the case where bytes are cut off the end of a zbuf, a value of **ZBUF\_NONE** is returned.

#### **VXWORKS AE PROTECTION DOMAINS**

Under VxWorks AE, you can call this function from within the kernel protection domain only. In addition, all arguments to this function can reference only that data which is valid in the kernel protection domain. Likewise, the returned value is valid in the protection domain only. This restriction does not apply under non-AE versions of VxWorks.

**RETURNS** The zbuf segment ID of the segment following the deleted bytes, or **NULL** if the operation fails.

**SEE ALSO** [zbufLib](#)

---

## **zbufDelete()**

**NAME** `zbufDelete()` – delete a zbuf

**SYNOPSIS**

```
STATUS zbufDelete
(
    ZBUF_ID zbufId           /* zbuf to be deleted */
)
```

**DESCRIPTION** This routine deletes any zbuf segments in the specified zbuf, then deletes the zbuf ID itself. *zbufId* must not be used after this routine executes successfully.

For any data buffers that were not in use by any other zbuf, **zbufDelete()** calls the associated free routine (callback).

#### **VXWORKS AE PROTECTION DOMAINS**

Under VxWorks AE, you can call this function from within the kernel protection domain only. In addition, all arguments to this function can reference only that data which is valid in the kernel protection domain. This restriction does not apply under non-AE versions of VxWorks.

**RETURNS** OK, or **ERROR** if the zbuf cannot be deleted.

**SEE ALSO** **zbufLib**, **zbufCreate()**, **zbufInsertBuf()**

---

## **zbufDup()**

**NAME** **zbufDup()** – duplicate a zbuf

**SYNOPSIS**

```
ZBUF_ID zbufDup  
(  
    ZBUF_ID zbufId,           /* zbuf to duplicate */  
    ZBUF_SEG zbufSeg,       /* zbuf segment base for offset */  
    int      offset,         /* relative byte offset */  
    int      len             /* number of bytes to duplicate */  
)
```

**DESCRIPTION** This routine duplicates *len* bytes of *zbufId* starting at the specified byte location, and returns the zbuf ID of the newly created duplicate zbuf.

The starting location of duplication is specified by *zbufSeg* and *offset*. See the **zbufLib** manual page for more information on specifying a byte location within a zbuf. In particular, the first byte duplicated is the exact byte specified by *zbufSeg* and *offset*.

The number of bytes to duplicate is given by *len*. If this parameter is negative, or is larger than the number of bytes in the zbuf after the specified byte location, the rest of the zbuf is duplicated.

Duplication of zbuf data does not usually involve copying of the data. Instead, the zbuf segment pointer information is duplicated, while the data is not, which means that the data is shared among all zbuf segments that reference the data. See the **zbufLib** manual page for more information on copying and sharing zbuf data.

**RETURNS** The zbuf ID of a newly created duplicate zbuf, or **NULL** if the operation fails.

**SEE ALSO** **zbufLib**

---

## zbufExtractCopy()

**NAME** zbufExtractCopy() – copy data from a zbuf to a buffer

**SYNOPSIS**

```
int zbufExtractCopy
(
    ZBUF_ID  zbufId,          /* zbuf from which data is copied */
    ZBUF_SEG zbufSeg,        /* zbuf segment base for offset */
    int      offset,         /* relative byte offset */
    caddr_t  buf,            /* buffer into which data is copied */
    int      len             /* number of bytes to copy */
)
```

**DESCRIPTION** This routine copies *len* bytes of data from *zbufId* to the application buffer *buf*.

The starting location of the copy is specified by *zbufSeg* and *offset*. See the **zbufLib** manual page for more information on specifying a byte location within a zbuf. In particular, the first byte copied is the exact byte specified by *zbufSeg* and *offset*.

The number of bytes to copy is given by *len*. If this parameter is negative, or is larger than the number of bytes in the zbuf after the specified byte location, the rest of the zbuf is copied. The bytes copied may span more than one segment.

### VXWORKS AE PROTECTION DOMAINS

Under VxWorks AE, you can call this function from within the kernel protection domain only. In addition, all arguments to this function can reference only that data which is valid in the kernel protection domain. This restriction does not apply under non-AE versions of VxWorks.

**RETURNS** The number of bytes copied from the zbuf to the buffer, or **ERROR** if the operation fails.

**SEE ALSO** zbufLib

---

## zbufInsert()

**NAME** zbufInsert() – insert a zbuf into another zbuf

**SYNOPSIS**

```
ZBUF_SEG zbufInsert
(
    ZBUF_ID zbufId1,          /* zbuf to insert zbufId2 into */
    ZBUF_SEG zbufSeg,        /* zbuf segment base for offset */
    int offset,              /* relative byte offset */
    ZBUF_ID zbufId2          /* zbuf to insert into zbufId1 */
)
```

**DESCRIPTION**

This routine inserts all *zbufId2* zbuf segments into *zbufId1* at the specified byte location. The location of insertion is specified by *zbufSeg* and *offset*. See the **zbufLib** manual page for more information on specifying a byte location within a zbuf. In particular, insertion within a zbuf occurs before the byte location specified by *zbufSeg* and *offset*. Additionally, *zbufSeg* and *offset* must be **NULL** and 0, respectively, when inserting into an empty zbuf.

After all the *zbufId2* segments are inserted into *zbufId1*, the zbuf ID *zbufId2* is deleted. *zbufId2* must not be used after this routine executes successfully.

### VXWORKS AE PROTECTION DOMAINS

Under VxWorks AE, you can call this function from within the kernel protection domain only. In addition, all arguments to this function can reference only that data which is valid in the kernel protection domain. Likewise, the returned **ZBUF\_SEG** is valid within the kernel protection domain only. This restriction does not apply under non-AE versions of VxWorks.

**RETURNS** The zbuf segment ID for the first inserted segment, or **NULL** if the operation fails.

**SEE ALSO** zbufLib

---

## zbufInsertBuf()

**NAME** `zbufInsertBuf()` – create a zbuf segment from a buffer and insert into a zbuf

**SYNOPSIS**

```
ZBUF_SEG zbufInsertBuf
(
    ZBUF_ID      zbufId,          /* zbuf in which buffer is inserted */
    ZBUF_SEG     zbufSeg,        /* zbuf segment base for offset */
    int          offset,         /* relative byte offset */
    caddr_t      buf,            /* application buffer for segment */
    int          len,            /* number of bytes to insert */
    VOIDFUNCPTR freeRtn,         /* free-routine callback */
    int          freeArg          /* argument to free routine */
)
```

**DESCRIPTION** This routine creates a zbuf segment from the application buffer *buf* and inserts it at the specified byte location in *zbufId*.

The location of insertion is specified by *zbufSeg* and *offset*. See the **zbufLib** manual page for more information on specifying a byte location within a zbuf. In particular, insertion within a zbuf occurs before the byte location specified by *zbufSeg* and *offset*. Additionally, *zbufSeg* and *offset* must be `NULL` and `0`, respectively, when inserting into an empty zbuf.

The parameter *freeRtn* specifies a free-routine callback that runs when the data buffer *buf* is no longer referenced by any zbuf segments. If *freeRtn* is `NULL`, the zbuf functions normally, except that the application is not notified when no more zbufs segments reference *buf*. The free-routine callback runs from the context of the task that last deletes reference to the buffer. Declare the *freeRtn* callback as follows (using whatever routine name suits your application):

```
void freeCallback
(
    caddr_t      buf,            /* pointer to application buffer */
    int          freeArg        /* argument to free routine */
)
```

### VXWORKS AE PROTECTION DOMAINS

Under VxWorks AE, you can call this function from within the kernel protection domain only. In addition, all arguments to this function can reference only that data which is valid in the kernel protection domain. This restriction does not apply under non-AE versions of VxWorks.

**RETURNS** The zbuf segment ID of the inserted segment, or `NULL` if the operation fails.

**SEE ALSO** `zbufLib`

## zbufInsertCopy()

**NAME** zbufInsertCopy() – copy buffer data into a zbuf

**SYNOPSIS**

```
ZBUF_SEG zbufInsertCopy
(
    ZBUF_ID zbufId,          /* zbuf into which data is copied */
    ZBUF_SEG zbufSeg,       /* zbuf segment base for offset */
    int offset,             /* relative byte offset */
    caddr_t buf,            /* buffer from which data is copied */
    int len                  /* number of bytes to copy */
)
```

**DESCRIPTION** This routine copies *len* bytes of data from the application buffer *buf* and inserts it at the specified byte location in *zbufId*. The application buffer is in no way tied to the zbuf after this operation; a separate copy of the data is made.

The location of insertion is specified by *zbufSeg* and *offset*. See the **zbufLib** manual page for more information on specifying a byte location within a zbuf. In particular, insertion within a zbuf occurs before the byte location specified by *zbufSeg* and *offset*. Additionally, *zbufSeg* and *offset* must be **NULL** and 0, respectively, when inserting into an empty zbuf.

### VXWORKS AE PROTECTION DOMAINS

Under VxWorks AE, you can call this function from within the kernel protection domain only. In addition, all arguments to this function can reference only that data which is valid in the kernel protection domain. Likewise, the returned value is valid in the protection domain only. This restriction does not apply under non-AE versions of VxWorks.

**RETURNS** The zbuf segment ID of the first inserted segment, or **NULL** if the operation fails.

**SEE ALSO** zbufLib

---

## zbufLength()

**NAME** zbufLength() – determine the length in bytes of a zbuf

**SYNOPSIS**

```
int zbufLength
(
    ZBUF_ID zbufId          /* zbuf to determine length */
)
```

**DESCRIPTION** This routine returns the number of bytes in the zbuf *zbufId*.

### VXWORKS AE PROTECTION DOMAINS

Under VxWorks AE, you can call this function from within the kernel protection domain only. In addition, all arguments to this function can reference only that data which is valid in the kernel protection domain. This restriction does not apply under non-AE versions of VxWorks.

**RETURNS** The number of bytes in the zbuf, or **ERROR** if the operation fails.

**SEE ALSO** zbufLib

---

## zbufSegData()

**NAME** zbufSegData() – determine the location of data in a zbuf segment

**SYNOPSIS**

```
caddr_t zbufSegData
(
    ZBUF_ID zbufId,        /* zbuf to examine */
    ZBUF_SEG zbufSeg      /* segment to get pointer to data */
)
```

**DESCRIPTION** This routine returns the location of the first byte of data in the zbuf segment *zbufSeg*. If *zbufSeg* is **NULL**, the location of data in the first segment in *zbufId* is returned.

### VXWORKS AE PROTECTION DOMAINS

Under VxWorks AE, you can call this function from within the kernel protection domain only. In addition, all arguments to this function can reference only that data which is valid in the kernel protection domain. Likewise, the returned value is valid in the protection domain only. This restriction does not apply under non-AE versions of VxWorks.

- RETURNS** A pointer to the first byte of data in the specified zbuf segment, or NULL if the operation fails.
- SEE ALSO** [zbufLib](#)

---

## zbufSegFind()

**NAME** `zbufSegFind()` – find the zbuf segment containing a specified byte location

**SYNOPSIS**

```
ZBUF_SEG zbufSegFind
(
    ZBUF_ID zbufId,          /* zbuf to examine */
    ZBUF_SEG zbufSeg,      /* zbuf segment base for pOffset */
    int * pOffset          /* relative byte offset */
)
```

**DESCRIPTION** This routine translates an address within a zbuf to its most local formulation. `zbufSegFind()` locates the zbuf segment in *zbufId* that contains the byte location specified by *zbufSeg* and *\*pOffset*, then returns that zbuf segment, and writes in *\*pOffset* the new offset relative to the returned segment.

If the *zbufSeg*, *\*pOffset* pair specify a byte location past the end of the zbuf, or before the first byte in the zbuf, `zbufSegFind()` returns NULL.

See the [zbufLib](#) manual page for a full discussion of addressing zbufs by segment and offset.

### VXWORKS AE PROTECTION DOMAINS

Under VxWorks AE, you can call this function from within the kernel protection domain only. In addition, all arguments to this function can reference only that data which is valid in the kernel protection domain. Likewise, the returned value is valid in the protection domain only. This restriction does not apply under non-AE versions of VxWorks.

**RETURNS** The zbuf segment ID of the segment containing the specified byte, or NULL if the operation fails.

**SEE ALSO** [zbufLib](#)

---

## zbufSegLength()

**NAME** zbufSegLength() – determine the length of a zbuf segment

**SYNOPSIS**

```
int zbufSegLength
(
    ZBUF_ID zbufId,          /* zbuf to examine */
    ZBUF_SEG zbufSeg        /* segment to determine length of */
)
```

**DESCRIPTION** This routine returns the number of bytes in the zbuf segment *zbufSeg*. If *zbufSeg* is NULL, the length of the first segment in *zbufId* is returned.

### VXWORKS AE PROTECTION DOMAINS

Under VxWorks AE, you can call this function from within the kernel protection domain only. In addition, all arguments to this function can reference only that data which is valid in the kernel protection domain. This restriction does not apply under non-AE versions of VxWorks.

**RETURNS** The number of bytes in the specified zbuf segment, or **ERROR** if the operation fails.

**SEE ALSO** zbufLib

---

## zbufSegNext()

**NAME** zbufSegNext() – get the next segment in a zbuf

**SYNOPSIS**

```
ZBUF_SEG zbufSegNext
(
    ZBUF_ID zbufId,          /* zbuf to examine */
    ZBUF_SEG zbufSeg        /* segment to get next segment */
)
```

**DESCRIPTION** This routine finds the zbuf segment in *zbufId* that is just after the zbuf segment *zbufSeg*. If *zbufSeg* is NULL, the segment after the first segment in *zbufId* is returned. If *zbufSeg* is the last segment in *zbufId*, NULL is returned.

**RETURNS** The zbuf segment ID of the segment after *zbufSeg*, or NULL if the operation fails.

**SEE ALSO** zbufLib

---

## zbufSegPrev()

**NAME** zbufSegPrev() – get the previous segment in a zbuf

**SYNOPSIS**

```
ZBUF_SEG zbufSegPrev
(
    ZBUF_ID zbufId,          /* zbuf to examine */
    ZBUF_SEG zbufSeg        /* segment to get previous segment */
)
```

**DESCRIPTION** This routine finds the zbuf segment in *zbufId* that is just before the zbuf segment *zbufSeg*. If *zbufSeg* is NULL, or is the first segment in *zbufId*, NULL is returned.

### VXWORKS AE PROTECTION DOMAINS

Under VxWorks AE, you can call this function from within the kernel protection domain only. In addition, all arguments to this function can reference only that data which is valid in the kernel protection domain. Likewise, the returned value is valid in the protection domain only. This restriction does not apply under non-AE versions of VxWorks.

**RETURNS** The zbuf segment ID of the segment before *zbufSeg*, or NULL if the operation fails.

**SEE ALSO** zbufLib

---

## zbufSockBufSend()

**NAME** zbufSockBufSend() – create a zbuf from user data and send it to a TCP socket

**SYNOPSIS**

```
int zbufSockBufSend
(
    int          s,          /* socket to send to */
    char *      buf,        /* pointer to data buffer */
    int         bufLen,     /* number of bytes to send */
    VOIDFUNCPTR freeRtn,    /* free routine callback */
    int         freeArg,    /* argument to free routine */
    int         flags       /* flags to underlying protocols */
)
```

**DESCRIPTION** This routine creates a zbuf from the user buffer *buf*, and transmits it to a previously established connection-based (stream) socket.

The user-provided free routine callback at *freeRtn* is called when *buf* is no longer in use by the TCP/IP network stack. Applications can exploit this callback to receive notification that *buf* is free. If *freeRtn* is **NULL**, the routine functions normally, except that the application has no way of being notified when *buf* is released by the network stack. The free routine runs in the context of the task that last references the buffer. This is typically either the context of **tNetTask**, or the context of the caller's task. Declare *freeRtn* as follows (using whatever name is convenient):

```
void freeCallback
(
    caddr_t    buf,    /* pointer to user buffer */
    int       freeArg /* user-provided argument to free routine */
)
```

You may OR the following values into the *flags* parameter with this operation:

**MSG\_OOB** (0x1)  
Out-of-band data.

**MSG\_DONTROUTE** (0x4)  
Send without using routing tables.

#### **VXWORKS AE PROTECTION DOMAINS**

Under VxWorks AE, you can call this function from within the kernel protection domain only. In addition, all arguments to this function can reference only that data which is valid in the kernel protection domain. This restriction does not apply under non-AE versions of VxWorks.

**RETURNS** The number of bytes sent, or **ERROR** if the call fails.

**SEE ALSO** **zbufSockLib**, **zbufSockSend()**, **send()**

---

## **zbufSockBufSendto()**

**NAME** **zbufSockBufSendto()** – create a zbuf from a user message and send it to a UDP socket

**SYNOPSIS**

```
int zbufSockBufSendto
(
    int          s,          /* socket to send to */
    char *      buf,        /* pointer to data buffer */
    int         bufLen,     /* number of bytes to send */
    VOIDFUNCPTR freeRtn,   /* free routine callback */
    int         freeArg,    /* argument to free routine */
    int         flags,      /* flags to underlying protocols */
)
```

```
    struct sockaddr * to,      /* recipient's address */  
    int             tolen     /* length of to socket addr */  
    )
```

**DESCRIPTION** This routine creates a zbuf from the user buffer *buf*, and sends it to the datagram socket named by *to*. The socket *s* is the sending socket.

The user-provided free routine callback at *freeRtn* is called when *buf* is no longer in use by the UDP/IP network stack. Applications can exploit this callback to receive notification that *buf* is free. If *freeRtn* is NULL, the routine functions normally, except that the application has no way of being notified when *buf* is released by the network stack. The free routine runs in the context of the task that last references the buffer. This is typically either **tNetTask** context, or the caller's task context. Declare *freeRtn* as follows (using whatever name is convenient):

```
    void freeCallback  
    (  
    caddr_t      buf,      /* pointer to user buffer */  
    int          freeArg /* user-provided argument to free routine */  
    )
```

You may OR the following values into the *flags* parameter with this operation:

**MSG\_OOB** (0x1)

Out-of-band data.

**MSG\_DONTROUTE** (0x4)

Send without using routing tables.

**VXWORKS AE PROTECTION DOMAINS**

Under VxWorks AE, you can call this function from within the kernel protection domain only. In addition, all arguments to this function can reference only that data which is valid in the kernel protection domain. This restriction does not apply under non-AE versions of VxWorks.

**RETURNS** The number of bytes sent, or **ERROR** if the call fails.

**SEE ALSO** **zbufSockLib**, **zbufSockSendto()**, **sendto()**

---

## zbufSockLibInit()

- NAME** `zbufSockLibInit()` – initialize the zbuf socket interface library
- SYNOPSIS** `STATUS zbufSockLibInit (void)`
- DESCRIPTION** This routine initializes the zbuf socket interface library. It must be called before any zbuf socket routines are used. It is called automatically when `INCLUDE_ZBUF_SOCKET` is defined.
- VXWORKS AE PROTECTION DOMAINS**  
Under VxWorks AE, you can call this function from within the kernel protection domain only. In addition, all arguments to this function can reference only that data which is valid in the kernel protection domain. This restriction does not apply under non-AE versions of VxWorks.
- RETURNS** `OK`, or `ERROR` if the zbuf socket interface could not be initialized.
- SEE ALSO** `zbufSockLib`

---

## zbufSockRecv()

- NAME** `zbufSockRecv()` – receive data in a zbuf from a TCP socket
- SYNOPSIS** `ZBUF_ID zbufSockRecv`  
(  
    **int** `s`,                            /\* socket to receive data from \*/  
    **int** `flags`,                       /\* flags to underlying protocols \*/  
    **int** \* `pLen`                       /\* number of bytes requested/returned \*/  
)
- DESCRIPTION** This routine receives data from a connection-based (stream) socket, and returns the data to the user in a newly created zbuf.
- The `pLen` parameter indicates the number of bytes requested by the caller. If the operation is successful, the number of bytes received is copied to `pLen`.
- You may OR the following values into the `flags` parameter with this operation:
- `MSG_OOB` (0x1)  
    Out-of-band data.

**MSG\_PEEK** (0x2)

Return data without removing it from socket.

Once the user application is finished with the zbuf, **zbufDelete()** should be called to return the zbuf memory buffer to the VxWorks network stack.

**VXWORKS AE PROTECTION DOMAINS**

Under VxWorks AE, you can call this function from within the kernel protection domain only. In addition, all arguments to this function can reference only that data which is valid in the kernel protection domain. This restriction does not apply under non-AE versions of VxWorks.

**RETURNS** The zbuf ID of a newly created zbuf containing the received data, or **NULL** if the operation fails.

**SEE ALSO** **zbufSockLib**, **recv()**

---

## **zbufSockRecvfrom()**

**NAME** **zbufSockRecvfrom()** – receive a message in a zbuf from a UDP socket

**SYNOPSIS**

```
ZBUF_ID zbufSockRecvfrom
(
    int          s,          /* socket to receive from */
    int          flags,     /* flags to underlying protocols */
    int *        pLen,      /* number of bytes requested/returned */
    struct sockaddr * from, /* where to copy sender's addr */
    int *        pFromLen /* value/result length of from */
)
```

**DESCRIPTION** This routine receives a message from a datagram socket, and returns the message to the user in a newly created zbuf.

The message is received regardless of whether the socket is connected. If *from* is nonzero, the address of the sender's socket is copied to it. Initialize the value-result parameter *pFromLen* to the size of the *from* buffer. On return, *pFromLen* contains the actual size of the address stored in *from*.

The *pLen* parameter indicates the number of bytes requested by the caller. If the operation is successful, the number of bytes received is copied to *pLen*.

You may OR the following values into the *flags* parameter with this operation:

**MSG\_OOB** (0x1)  
Out-of-band data.

**MSG\_PEEK** (0x2)  
Return data without removing it from socket.

Once the user application is finished with the zbuf, **zbufDelete()** should be called to return the zbuf memory buffer to the VxWorks network stack.

#### VXWORKS AE PROTECTION DOMAINS

Under VxWorks AE, you can call this function from within the kernel protection domain only. In addition, all arguments to this function can reference only that data which is valid in the kernel protection domain. This restriction does not apply under non-AE versions of VxWorks.

**RETURNS** The zbuf ID of a newly created zbuf containing the received message, or NULL if the operation fails.

**SEE ALSO** **zbufSockLib**

---

## zbufSockSend()

**NAME** **zbufSockSend()** – send zbuf data to a TCP socket

**SYNOPSIS**

```
int zbufSockSend
(
    int      s,                /* socket to send to */
    ZBUF_ID zbufId,           /* zbuf to transmit */
    int      zbufLen,         /* length of entire zbuf */
    int      flags             /* flags to underlying protocols */
)
```

**DESCRIPTION** This routine transmits all of the data in *zbufId* to a previously established connection-based (stream) socket.

The *zbufLen* parameter is used only for determining the amount of space needed from the socket write buffer. *zbufLen* has no effect on how many bytes are sent; the entire zbuf is always transmitted. If the length of *zbufId* is not known, the caller must first determine it by calling **zbufLength()**.

This routine transfers ownership of the zbuf from the user application to the VxWorks network stack. The zbuf ID, *zbufId*, is deleted by this routine, and should not be used after the routine is called, even if an **ERROR** status is returned. (Exceptions: when the routine fails because the zbuf socket interface library was not initialized or an invalid zbuf ID was

passed in, in which case there is no zbuf to delete. Moreover, if the call fails during a non-blocking I/O socket write with an **errno** of **EWOULDBLOCK**, then *zbufId* is not deleted; thus the caller may send it again at a later time.)

You may OR the following values into the *flags* parameter with this operation:

**MSG\_OOB** (0x1)

Out-of-band data.

**MSG\_DONTROUTE** (0x4)

Send without using routing tables.

#### **VXWORKS AE PROTECTION DOMAINS**

Under VxWorks AE, you can call this function from within the kernel protection domain only. In addition, all arguments to this function can reference only that data which is valid in the kernel protection domain. This restriction does not apply under non-AE versions of VxWorks.

**RETURNS** The number of bytes sent, or **ERROR** if the call fails.

**SEE ALSO** **zbufSockLib**, **zbufLength( )**, **zbufSockBufSend( )**, **send( )**

---

## **zbufSockSendto( )**

**NAME** **zbufSockSendto( )** – send a zbuf message to a UDP socket

**SYNOPSIS**

```
int zbufSockSendto
(
    int          s,          /* socket to send to */
    ZBUF_ID     zbufId,     /* zbuf to transmit */
    int         zbufLen,    /* length of entire zbuf */
    int         flags,      /* flags to underlying protocols */
    struct sockaddr * to,   /* recipient's address */
    int         tolen       /* length of to socket addr */
)
```

**DESCRIPTION** This routine sends the entire message in *zbufId* to the datagram socket named by *to*. The socket *s* is the sending socket.

The *zbufLen* parameter is used only for determining the amount of space needed from the socket write buffer. *zbufLen* has no effect on how many bytes are sent; the entire zbuf is always transmitted. If the length of *zbufId* is not known, the caller must first determine it by calling **zbufLength( )**.

This routine transfers ownership of the zbuf from the user application to the VxWorks network stack. The zbuf ID *zbufId* is deleted by this routine, and should not be used after the routine is called, even if an **ERROR** status is returned. (Exceptions: when the routine fails because the zbuf socket interface library was not initialized or an invalid zbuf ID was passed in, in which case there is no zbuf to delete. Moreover, if the call fails during a non-blocking I/O socket write with an **errno** of **EWOULDBLOCK**, then *zbufId* is not deleted; thus the caller may send it again at a later time.)

You may OR the following values into the *flags* parameter with this operation:

**MSG\_OOB** (0x1)  
Out-of-band data.

**MSG\_DONTROUTE** (0x4)  
Send without using routing tables.

#### VXWORKS AE PROTECTION DOMAINS

Under VxWorks AE, you can call this function from within the kernel protection domain only. In addition, all arguments to this function can reference only that data which is valid in the kernel protection domain. This restriction does not apply under non-AE versions of VxWorks.

**RETURNS** The number of bytes sent, or **ERROR** if the call fails.

**SEE ALSO** `zbufSockLib`, `zbufLength()`, `zbufSockBufSendto()`, `sendto()`

---

## zbufSplit()

**NAME** `zbufSplit()` – split a zbuf into two separate zbufs

**SYNOPSIS**

```
ZBUF_ID zbufSplit
(
    ZBUF_ID zbufId,           /* zbuf to split into two */
    ZBUF_SEG zbufSeg,       /* zbuf segment base for offset */
    int      offset         /* relative byte offset */
)
```

**DESCRIPTION** This routine splits *zbufId* into two separate zbufs at the specified byte location. The first portion remains in *zbufId*, while the end portion is returned in a newly created zbuf.

The location of the split is specified by *zbufSeg* and *offset*. See the **zbufLib** manual page for more information on specifying a byte location within a zbuf. In particular, after the split operation, the first byte of the returned zbuf is the exact byte specified by *zbufSeg* and *offset*.

**VXWORKS AE PROTECTION DOMAINS**

Under VxWorks AE, you can call this function from within the kernel protection domain only. In addition, all arguments to this function can reference only that data which is valid in the kernel protection domain. Likewise, the returned value is valid in the protection domain only. This restriction does not apply under non-AE versions of VxWorks.

**RETURNS** The zbuf ID of a newly created zbuf containing the end portion of *zbufId*, or **NULL** if the operation fails.

**SEE ALSO** [zbufLib](#)

# Keyword Index

	Keyword	Name	Page
interfaces. POSIX	1003.1c thread library.....	<b>pthreadLib</b>	217
compare keys as	32 bit identifiers. ....	<b>hashKeyCmp()</b>	645
library for PentiumPro/2/3/4	32 bit mode. MMU .....	<b>mmuPro32Lib</b>	163
read next word	(32-bit integer) from stream. ....	<b>getw()</b>	640
write word	(32-bit integer) to stream. ....	<b>putw()</b>	1047
exception vector (PowerPC	403). /C routine to critical.....	<b>excCrtConnect()</b>	578
interrupt vector (PowerPC	403). /C routine to critical.....	<b>excIntCrtConnect()</b>	581
initialize	4kc cache library. ....	<b>cache4kcLibInit()</b>	445
MIPS	4kc cache management library. ....	<b>cache4kcLib</b>	37
get lower half of	64Bit TSC (Timestamp Counter). ....	<b>pentiumTscGet32()</b>	974
get	64Bit TSC (Timestamp Counter). ....	<b>pentiumTscGet64()</b>	974
of register a0 (also a1 - a7)	(68K). return contents .....	<b>a0()</b>	403
of register d0 (also d1 - d7)	(68K). return contents .....	<b>d0()</b>	512
set task status register	(68K, MIPS, x86). ....	<b>taskSRSet()</b>	1347
contents of status register	(68K, SH). return.....	<b>sr()</b>	1262
clear entry from cache	(68K, x86). ....	<b>cacheArchClearEntry()</b>	446
interrupt lock-out level	(68K, x86, ARM, SH, / /current.....	<b>intLockLevelGet()</b>	693
interrupt lock-out level	(68K, x86, ARM, SH, / /current.....	<b>intLockLevelSet()</b>	693
/exception vector table	(68K, x86, ARM, SimSolaris, / .....	<b>intVecTableWriteProtect()</b>	704
SimNT and/ set interrupt level	(68K, x86, ARM, SimSolaris, .....	<b>intLevelSet()</b>	690
get vector (trap) base address	(68K, x86, MIPS, ARM, / .....	<b>intVecBaseGet()</b>	696
set vector (trap) base address	(68K, x86, MIPS, ARM, / .....	<b>intVecBaseSet()</b>	697
get interrupt vector	(68K, x86, MIPS, SH, / .....	<b>intVecGet()</b>	698
set CPU vector (trap)	(68K, x86, MIPS, SH, / .....	<b>intVecSet()</b>	699
/handler for C routine	(68K, x86, MIPS, SimSolaris).....	<b>intHandlerCreate()</b>	688
increment packet counters for	802.3 device. ....	<b>m2If8023PacketCount()</b>	772
system library. ISO	9660 CD-ROM read-only file .....	<b>cdromFsLib</b>	59
return contents of register	a0 (also a1 - a7) (68K). ....	<b>a0()</b>	403
contents of register a0 (also	a1 - a7) (68K). return .....	<b>a0()</b>	403
of register a0 (also a1 -	a7) (68K). return contents.....	<b>a0()</b>	403
change	abort character. ....	<b>tyAbortSet()</b>	1406

	Keyword	Name	Page
	set abort function.....	<b>tyAbortFuncSet()</b>	1406
	compute absolute value (ANSI).....	<b>fabs()</b>	586
	compute absolute value (ANSI).....	<b>fabsf()</b>	586
	(ANSI). compute absolute value of integer.....	<b>abs()</b>	404
	compute absolute value of long (ANSI).....	<b>labs()</b>	729
	accept connection from socket.....	<b>accept()</b>	404
	acknowledge bus interrupt.....	<b>sysBusIntAck()</b>	1300
Internet protocol/	display all active connections for.....	<b>inetstatShow()</b>	681
send advertisement to all	active locations.....	<b>sendAdvertAll()</b>	1197
get list of	active task IDs.....	<b>taskIdListGet()</b>	1332
close	active telnet session.....	<b>telnetdExit()</b>	1359
start collecting task	activity data.....	<b>spyClkStart()</b>	1257
stop collecting task	activity data.....	<b>spyClkStop()</b>	1258
display task	activity data.....	<b>spyReport()</b>	1259
spy CPU	activity library.....	<b>spyLib</b>	294
begin periodic task	activity reports.....	<b>spy()</b>	1257
run periodic task	activity reports.....	<b>spyTask()</b>	1260
distributed objects interface	adapter show routines/.....	<b>distIfShow</b>	83
/about installed interface	adapter (VxFusion).....	<b>distIfShow()</b>	548
to handle Ethernet multicast	addresses. library.....	<b>etherMultiLib</b>	104
retrieve table of multicast	addresses from driver.....	<b>etherMultiGet()</b>	574
packet. get	addressing information from.....	<b>muxPacketAddrGet()</b>	885
bytes.	advance ring pointer by n.....	<b>rngMoveAhead()</b>	1112
locations. send	advertisement to all active.....	<b>sendAdvertAll()</b>	1197
send	advertisement to one location.....	<b>sendAdvert()</b>	1196
routines to RIP for SNMP	Agent. VxWorks interface.....	<b>m2RipLib</b>	147
library. WDB	agent context management.....	<b>wdbLib</b>	348
DHCP relay	agent library.....	<b>dhcprLib</b>	75
MIB-II ICMP-group API for SNMP	agents.....	<b>m2IcmpLib</b>	139
interface-group API for SNMP	agents. MIB-II.....	<b>m2IfLib</b>	139
MIB-II IP-group API for SNMP	agents.....	<b>m2IpLib</b>	142
MIB-II API library for SNMP	agents.....	<b>m2Lib</b>	144
system-group API for SNMP	agents. MIB-II.....	<b>m2SysLib</b>	148
MIB-II TCP-group API for SNMP	agents.....	<b>m2TcpLib</b>	150
MIB-II UDP-group API for SNMP	agents.....	<b>m2UdpLib</b>	152
initialize asynchronous I/O	(AIO) library.....	<b>aioPxLibInit()</b>	406
asynchronous I/O	(AIO) library (POSIX).....	<b>aioPxLib</b>	9
show	AIO requests.....	<b>aioShow()</b>	407
asynchronous I/O	(AIO) show library.....	<b>aioPxShow</b>	13
	AIO system driver.....	<b>aioSysDrv</b>	13
initialize	AIO system driver.....	<b>aioSysInit()</b>	407
signal. set	alarm clock for delivery of.....	<b>alarm()</b>	411
library.	Alchemy Au cache management.....	<b>cacheAuLib</b>	38
that are not necessarily	aligned. /bytes with buffers.....	<b>uswab()</b>	1426
allocate	aligned memory.....	<b>memalign()</b>	806
allocate	aligned memory from partition.....	<b>memPartAlignedAlloc()</b>	813
	allocate aligned memory.....	<b>memalign()</b>	806
partition.	allocate aligned memory from.....	<b>memPartAlignedAlloc()</b>	813
partition.	allocate block of memory from.....	<b>memPartAlloc()</b>	814
shared memory system/	allocate block of memory from.....	<b>smMemMalloc()</b>	1234

	Keyword	Name	Page
system memory partition/ DMA devices and drivers.	allocate block of memory from.....	<b>malloc()</b>	802
possible.	allocate cache-safe buffer for .....	<b>cacheDmaMalloc()</b>	453
shared memory system/ boundary.	allocate cache-safe buffer, if .....	<b>cacheR32kMalloc()</b>	464
(ANSI).	allocate memory for array from .....	<b>smMemCalloc()</b>	1232
interface table.	allocate memory on page .....	<b>valloc()</b>	1429
pool of buffers (VxFusion).	allocate space for array .....	<b>calloc()</b>	476
clock for timing base/ (C++). call	allocate structure for .....	<b>m2IfAlloc()</b>	773
test whether character is	allocate telegram buffer from .....	<b>distTBufAlloc()</b>	556
write character to stream	allocate timer using specified .....	<b>timer_create()</b>	1382
to standard output stream	allocation failure handler .....	<b>cplusCallNewHandler()</b>	502
to standard output stream	alphanumeric (ANSI).....	<b>isalnum()</b>	719
sort array of objects	announce clock tick to kernel. ....	<b>tickAnnounce()</b>	1379
between 0 and RAND_MAX	(ANSI). .....	<b>putc()</b>	1045
reallocate block of memory	(ANSI). write character.....	<b>putchar()</b>	1046
remove file	(ANSI). write string.....	<b>puts()</b>	1047
indicator to beginning of file	(ANSI). .....	<b>qsort()</b>	1049
from standard input stream	(ANSI). /pseudo-random integer.....	<b>rand()</b>	1054
specify buffering for stream	(ANSI). .....	<b>realloc()</b>	1065
in jmp_buf argument	(ANSI). .....	<b>remove()</b>	1071
set appropriate locale	(ANSI). set file position .....	<b>rewind()</b>	1082
specify buffering for stream	(ANSI). /convert characters .....	<b>scanf()</b>	1130
compute sine	(ANSI). .....	<b>setbuf()</b>	1200
compute sine	(ANSI). /calling environment.....	<b>setjmp()</b>	1201
compute hyperbolic sine	(ANSI). .....	<b>setlocale()</b>	1203
compute hyperbolic sine	(ANSI). .....	<b>setvbuf()</b>	1211
formatted string to buffer	(ANSI). .....	<b>sin()</b>	1228
non-negative square root	(ANSI). .....	<b>sinf()</b>	1229
non-negative square root	(ANSI). .....	<b>sinh()</b>	1230
to generate random numbers	(ANSI). .....	<b>sinhf()</b>	1230
characters from ASCII string	(ANSI). write.....	<b>sprintf()</b>	1256
one string to another	(ANSI). compute.....	<b>sqrt()</b>	1261
of character in string	(ANSI). compute.....	<b>sqrtf()</b>	1261
two strings lexicographically	(ANSI). /value of seed used.....	<b>srand()</b>	1262
as appropriate to LC_COLLATE	(ANSI). read and convert .....	<b>sscanf()</b>	1263
copy one string to another	(ANSI). concatenate .....	<b>strcat()</b>	1271
first character from given set	(ANSI). find first occurrence.....	<b>strchr()</b>	1271
error number to error string	(ANSI). compare.....	<b>strcmp()</b>	1272
time into formatted string	(ANSI). compare two strings.....	<b>strcoll()</b>	1272
determine length of string	(ANSI). .....	<b>strcpy()</b>	1273
from one string to another	(ANSI). /string length up to.....	<b>strcspn()</b>	1273
n characters of two strings	(ANSI). map .....	<b>strerror()</b>	1274
from one string to another	(ANSI). convert broken-down.....	<b>strftime()</b>	1275
of character from given set	(ANSI). .....	<b>strlen()</b>	1277
of character in string	(ANSI). /characters.....	<b>strncat()</b>	1277
character not in given set	(ANSI). compare first.....	<b>strncmp()</b>	1278
of substring in string	(ANSI). copy characters.....	<b>strncpy()</b>	1278
	(ANSI). /occurrence in string.....	<b>strpbrk()</b>	1279
	(ANSI). find last occurrence.....	<b>strrchr()</b>	1279
	(ANSI). /length up to first .....	<b>strspn()</b>	1280
	(ANSI). find first occurrence.....	<b>strstr()</b>	1280

	Keyword	Name	Page
portion of string to double	(ANSI). convert initial .....	<b>strtod()</b>	1281
break down string into tokens	(ANSI).....	<b>strtok()</b>	1282
convert string to long integer	(ANSI).....	<b>strtol()</b>	1284
to unsigned long integer	(ANSI). convert string .....	<b>strtolu()</b>	1285
to n characters of s2 into s1	(ANSI). transform up .....	<b>strxfrm()</b>	1287
processor (Unimplemented)	(ANSI). /string to command.....	<b>system()</b>	1317
compute tangent	(ANSI).....	<b>tan()</b>	1318
compute tangent	(ANSI).....	<b>tanf()</b>	1318
compute hyperbolic tangent	(ANSI).....	<b>tanh()</b>	1319
compute hyperbolic tangent	(ANSI).....	<b>tanhf()</b>	1319
current calendar time	(ANSI). determine.....	<b>time()</b>	1380
binary file (Unimplemented)	(ANSI). create temporary.....	<b>tmpfile()</b>	1391
generate temporary file name	(ANSI).....	<b>tmpnam()</b>	1391
to lower-case equivalent	(ANSI). /upper-case letter.....	<b>tolower()</b>	1392
to upper-case equivalent	(ANSI). /lower-case letter .....	<b>toupper()</b>	1392
back into input stream	(ANSI). push character.....	<b>ungetc()</b>	1413
formatted string to stream	(ANSI). write .....	<b>fprintf()</b>	1431
list to standard output	(ANSI). /variable argument.....	<b>fprintf()</b>	1446
argument list to buffer	(ANSI). /with variable.....	<b>vsprintf()</b>	1446
char's (Unimplemented)	(ANSI). /char's to multibyte.....	<b>wcstombs()</b>	1461
character (Unimplemented)	(ANSI). /to multibyte.....	<b>wctomb()</b>	1461
abnormal program termination	(ANSI). cause .....	<b>abort()</b>	403
absolute value of integer	(ANSI). compute .....	<b>abs()</b>	404
compute arc cosine	(ANSI).....	<b>acos()</b>	405
compute arc cosine	(ANSI).....	<b>acosf()</b>	406
broken-down time into string	(ANSI). convert .....	<b>asctime()</b>	416
compute arc sine	(ANSI).....	<b>asin()</b>	417
compute arc sine	(ANSI).....	<b>asinf()</b>	417
put diagnostics into programs	(ANSI).....	<b>assert()</b>	418
compute arc tangent	(ANSI).....	<b>atan()</b>	418
compute arc tangent of y/x	(ANSI).....	<b>atan2()</b>	419
compute arc tangent of y/x	(ANSI).....	<b>atan2f()</b>	420
compute arc tangent	(ANSI).....	<b>atanf()</b>	420
termination (Unimplemented)	(ANSI). /function at program.....	<b>atexit()</b>	421
convert string to double	(ANSI).....	<b>atof()</b>	421
convert string to int	(ANSI).....	<b>atoi()</b>	422
convert string to long	(ANSI).....	<b>atol()</b>	422
perform binary search	(ANSI).....	<b>bsearch()</b>	443
allocate space for array	(ANSI).....	<b>calloc()</b>	476
or equal to specified value	(ANSI). /integer greater than .....	<b>ceil()</b>	489
or equal to specified value	(ANSI). /integer greater than .....	<b>ceilf()</b>	490
and error flags for stream	(ANSI). clear end-of-file .....	<b>clearerr()</b>	493
processor time in use	(ANSI). determine.....	<b>clock()</b>	494
compute cosine	(ANSI).....	<b>cos()</b>	500
compute cosine	(ANSI).....	<b>cosf()</b>	500
compute hyperbolic cosine	(ANSI).....	<b>cosh()</b>	501
compute hyperbolic cosine	(ANSI).....	<b>coshf()</b>	501
time in seconds into string	(ANSI). convert .....	<b>ctime()</b>	510
between two calendar times	(ANSI). compute difference.....	<b>difftime()</b>	542
compute quotient and remainder	(ANSI).....	<b>div()</b>	557

	Keyword	Name	Page
	exit task (ANSI).....	<b>exit()</b>	584
	compute exponential value (ANSI).....	<b>exp()</b>	585
	compute exponential value (ANSI).....	<b>expf()</b>	585
	compute absolute value (ANSI).....	<b>fabs()</b>	586
	compute absolute value (ANSI).....	<b>fabsf()</b>	586
	close stream (ANSI).....	<b>fclose()</b>	587
	indicator for stream (ANSI). test end-of-file.....	<b>feof()</b>	588
	indicator for file pointer (ANSI). test error.....	<b>ferror()</b>	589
	flush stream (ANSI).....	<b>fflush()</b>	589
	next character from stream (ANSI). return.....	<b>fgetc()</b>	590
	position indicator for stream (ANSI). /current value of file.....	<b>fgetpos()</b>	590
	of characters from stream (ANSI). read specified number.....	<b>fgets()</b>	591
	or equal to specified value (ANSI). /integer less than.....	<b>floor()</b>	597
	or equal to specified value (ANSI). /integer less than.....	<b>floorf()</b>	597
	compute remainder of x/y (ANSI).....	<b>fmod()</b>	598
	compute remainder of x/y (ANSI).....	<b>fmodf()</b>	598
	open file specified by name (ANSI).....	<b>fopen()</b>	599
	formatted string to stream (ANSI). write.....	<b>fprintf()</b>	606
	write character to stream (ANSI).....	<b>fputc()</b>	610
	write string to stream (ANSI).....	<b>fputs()</b>	611
	read data into array (ANSI).....	<b>fread()</b>	611
	free block of memory (ANSI).....	<b>free()</b>	612
	open file specified by name (ANSI).....	<b>freopen()</b>	612
	fraction and power of 2 (ANSI). /into normalized.....	<b>frexp()</b>	613
	convert characters from stream (ANSI). read and.....	<b>fscanf()</b>	614
	position indicator for stream (ANSI). set file.....	<b>fseek()</b>	618
	position indicator for stream (ANSI). set file.....	<b>fsetpos()</b>	619
	position indicator for stream (ANSI). /current value of file.....	<b>ftell()</b>	620
	write from specified array (ANSI).....	<b>fwrite()</b>	633
	next character from stream (ANSI). return.....	<b>getc()</b>	635
	from standard input stream (ANSI). return next character.....	<b>getchar()</b>	635
	get environment variable (ANSI).....	<b>getenv()</b>	636
	from standard input stream (ANSI). read characters.....	<b>gets()</b>	638
	time into UTC broken-down time (ANSI). convert calendar.....	<b>gmtime()</b>	641
	character is alphanumeric (ANSI). test whether.....	<b>isalnum()</b>	719
	whether character is letter (ANSI). test.....	<b>isalpha()</b>	720
	character is control character (ANSI). test whether.....	<b>iscntrl()</b>	721
	character is decimal digit (ANSI). test whether.....	<b>isdigit()</b>	721
	non-white-space character (ANSI). /is printing,.....	<b>isgraph()</b>	722
	character is lower-case letter (ANSI). test whether.....	<b>islower()</b>	722
	including space character (ANSI). /is printable,.....	<b>isprint()</b>	723
	character is punctuation (ANSI). test whether.....	<b>ispunct()</b>	723
	is white-space character (ANSI). /whether character.....	<b>isspace()</b>	724
	character is upper-case letter (ANSI). test whether.....	<b>isupper()</b>	724
	character is hexadecimal digit (ANSI). test whether.....	<b>isxdigit()</b>	725
	compute absolute value of long (ANSI).....	<b>labs()</b>	729
	number by integral power of 2 (ANSI). multiply.....	<b>ldexp()</b>	731
	and remainder of division (ANSI). compute quotient.....	<b>ldiv()</b>	732
	of object with type lconv (ANSI). set components.....	<b>localeconv()</b>	742
	time into broken-down time (ANSI). convert calendar.....	<b>localtime()</b>	745

	Keyword	Name	Page
compute natural logarithm	(ANSI).....	<b>log()</b>	746
compute base-10 logarithm	(ANSI).....	<b>log10()</b>	748
compute base-10 logarithm	(ANSI).....	<b>log10f()</b>	748
compute natural logarithm	(ANSI).....	<b>logf()</b>	749
by restoring saved environment	(ANSI). /non-local goto.....	<b>longjmp()</b>	759
from system memory partition	(ANSI). /block of memory.....	<b>malloc()</b>	802
character (Unimplemented)	(ANSI). /length of multibyte.....	<b>mblen()</b>	804
to wide char's (Unimplemented)	(ANSI). /of multibyte char's.....	<b>mbstowcs()</b>	804
wide character (Unimplemented)	(ANSI). /character to.....	<b>mbtowc()</b>	805
block of memory for character	(ANSI). search.....	<b>memchr()</b>	807
compare two blocks of memory	(ANSI).....	<b>memcmp()</b>	807
from one location to another	(ANSI). copy memory.....	<b>memcpy()</b>	808
from one location to another	(ANSI). copy memory.....	<b>memmove()</b>	812
set block of memory	(ANSI).....	<b>memset()</b>	819
time into calendar time	(ANSI). convert broken-down.....	<b>mktime()</b>	822
integer and fraction parts	(ANSI). /number into.....	<b>modf()</b>	827
in errno to error message	(ANSI). map error number.....	<b> perror()</b>	976
raised to specified power	(ANSI). /value of number.....	<b>pow()</b>	981
raised to specified power	(ANSI). /value of number.....	<b>powf()</b>	982
to standard output stream	(ANSI). /formatted string.....	<b>printf()</b>	997
	ANSI assert documentation.....	<b>ansiAssert</b>	13
	ANSI ctype documentation.....	<b>ansiCtype</b>	14
	ANSI locale documentation.....	<b>ansiLocale</b>	15
	ANSI math documentation.....	<b>ansiMath</b>	15
	ANSI setjmp documentation.....	<b>ansiSetjmp</b>	16
	ANSI stdarg documentation.....	<b>ansiStdarg</b>	17
	ANSI stdio documentation.....	<b>ansiStdio</b>	18
	ANSI stdlib documentation.....	<b>ansiStdlib</b>	22
	ANSI string documentation.....	<b>ansiString</b>	24
	ANSI time documentation.....	<b>ansiTime</b>	25
host responses. set	applette to stop FTP transient.....	<b>ftpTransientFatalInstall()</b>	631
exists. install	applette to test if file.....	<b>netDrvFileDoesNotExistInstall()</b>	914
compute	arc cosine (ANSI).....	<b>acos()</b>	405
compute	arc cosine (ANSI).....	<b>acosf()</b>	406
compute	arc sine (ANSI).....	<b>asin()</b>	417
compute	arc sine (ANSI).....	<b>asinf()</b>	417
compute	arc tangent (ANSI).....	<b>atan()</b>	418
compute	arc tangent (ANSI).....	<b>atanf()</b>	420
compute	arc tangent of y/x (ANSI).....	<b>atan2()</b>	419
compute	arc tangent of y/x (ANSI).....	<b>atan2f()</b>	420
cache library for processor	architecture. initialize.....	<b>cacheLibInit()</b>	457
MCA (Machine Check	Architecture). enable/disable.....	<b>pentiumMcaEnable()</b>	953
show MCA (Machine Check	Architecture) registers.....	<b>pentiumMcaShow()</b>	953
debugger library.	architecture-dependent.....	<b>dbgArchLib</b>	63
floating-point coprocessor/	architecture-dependent.....	<b>fppArchLib</b>	109
interrupt library.	architecture-dependent.....	<b>intArchLib</b>	123
block size (VxVMI). get	architecture-dependent page.....	<b>vmPageBlockSizeGet()</b>	1441
interrupt subroutine library.	architecture-independent.....	<b>intLib</b>	125
virtual memory support/	architecture-independent.....	<b>vmLib</b>	343
management library.	architecture-specific cache.....	<b>cacheArchLib</b>	37

	Keyword	Name	Page
exception-handling/	architecture-specific .....	<b>excArchLib</b>	105
<b>vxMemProbe()</b>	architecture-specific part of.....	<b>vxMemArchProbe()</b>	1454
management routines.	architecture-specific task .....	<b>taskArchLib</b>	307
tape in tar format.	archive named file/dir onto .....	<b>tarArchive()</b>	1322
psr value, symbolically	(ARM). /meaning of specified .....	<b>psrShow()</b>	1006
processor status register	(ARM). /contents of current.....	<b>cpsr()</b>	508
exception vector (PowerPC,	ARM). /routine to asynchronous .....	<b>excIntConnect()</b>	580
CPU exception vector (PowerPC,	ARM). get .....	<b>excVecGet()</b>	582
CPU exception vector (PowerPC,	ARM). set.....	<b>excVecSet()</b>	584
interrupt bits (MIPS, PowerPC,	ARM). disable corresponding .....	<b>intDisable()</b>	687
interrupt bits (MIPS, PowerPC,	ARM). enable corresponding.....	<b>intEnable()</b>	688
uninitialized vector handler	(ARM). set.....	<b>intUninitVecSet()</b>	695
address to virtual address	(ARM). translate physical .....	<b>mmuPhysToVirt()</b>	823
address to physical address	(ARM). translate virtual .....	<b>mmuVirtToPhys()</b>	826
MMU mapping library for	ARM Ltd. processors. ....	<b>mmuMapLib</b>	162
(also r1 - r14, r1-r15 for SH)	(ARM, SH). /of register r0 .....	<b>r0()</b>	1050
/lock-out level (68K, x86,	ARM, SH, SimSolaris, SimNT). ....	<b>intLockLevelGet()</b>	693
/lock-out level (68K, x86,	ARM, SH, SimSolaris, SimNT). ....	<b>intLockLevelSet()</b>	693
/base address (68K, x86, MIPS,	ARM, SimSolaris, SimNT).....	<b>intVecBaseGet()</b>	696
/base address (68K, x86, MIPS,	ARM, SimSolaris, SimNT).....	<b>intVecBaseSet()</b>	697
/vector table (68K, x86,	ARM, SimSolaris, SimNT).....	<b>intVecTableWriteProtect()</b>	704
set interrupt level (68K, x86,	ARM, SimSolaris, SimNT and/ .....	<b>intLevelSet()</b>	690
time until next expiration and	arm timer (POSIX). set.....	<b>timer_settime()</b>	1384
initialize proxy	ARP.....	<b>proxyArpLibInit()</b>	1001
Address Resolution Protocol	(ARP) client library. proxy .....	<b>proxyLib</b>	216
display known	ARP entries.....	<b>arptabShow()</b>	415
add, modify, or delete MIB-II	ARP entry.....	<b>m2IpAtransTblEntrySet()</b>	788
create proxy	ARP network.....	<b>proxyNetCreate()</b>	1002
show proxy	ARP networks.....	<b>proxyNetShow()</b>	1003
Address Resolution Protocol	(ARP) server library. proxy .....	<b>proxyArpLib</b>	215
flush all entries in system	ARP table.....	<b>arpFlush()</b>	413
display entries in system	ARP table.....	<b>arpShow()</b>	415
create or modify	ARP table entry.....	<b>arpAdd()</b>	412
remove	ARP table entry.....	<b>arpDelete()</b>	413
get MIB-II	ARP table entry.....	<b>m2IpAtransTblEntryGet()</b>	787
Address Resolution Protocol	(ARP) table manipulation/ .....	<b>arpLib</b>	26
allocate space for	array (ANSI).....	<b>calloc()</b>	476
read data into	array (ANSI).....	<b>fread()</b>	611
write from specified	array (ANSI).....	<b>fwrite()</b>	633
system/ allocate memory for	array from shared memory .....	<b>smMemCalloc()</b>	1232
sort	array of objects (ANSI). ....	<b>qsort()</b>	1049
and convert characters from	ASCII string (ANSI). read .....	<b>sscanf()</b>	1263
ANSI	assert documentation.....	<b>ansiAssert</b>	13
(Western Digital WD33C93/	assert RST line on SCSI bus .....	<b>sysScsiBusReset()</b>	1312
connect C routine to	asynchronous exception vector/ .....	<b>excIntConnect()</b>	580
library. initialize	asynchronous I/O (AIO).....	<b>aioPxLibInit()</b>	406
(POSIX).	asynchronous I/O (AIO) library .....	<b>aioPxLib</b>	9
library.	asynchronous I/O (AIO) show .....	<b>aioPxShow</b>	13
retrieve error status of	asynchronous I/O operation/.....	<b>aio_error()</b>	408
retrieve return status of	asynchronous I/O operation/ .....	<b>aio_return()</b>	409

	Keyword	Name	Page
(POSIX). wait for	asynchronous I/O request(s) .....	<b>aio_suspend()</b>	410
(POSIX). initiate list of	asynchronous I/O requests .....	<b>lio_listio()</b>	735
initiate	asynchronous read (POSIX) .....	<b>aio_read()</b>	408
initiate	asynchronous write (POSIX) .....	<b>aio_write()</b>	410
mount DOS file system from	ATA hard disk or CDROM .....	<b>usrAtaConfig()</b>	1419
	ATA/ATAPI initialization .....	<b>usrAta</b>	335
instruction to clear/ execute	atomic compare-and-exchange .....	<b>pentiumBtc()</b>	951
instruction to set/ execute	atomic compare-and-exchange .....	<b>pentiumBts()</b>	951
C-callable	atomic test-and-set primitive .....	<b>vxTas()</b>	1459
get value of prioceiling	attr in mutex attr object/ ....	<b>pthread_mutexattr_getprioceiling()</b>	1035
object/ set prioceiling	attr in mutex attributes .....	<b>pthread_mutexattr_setprioceiling()</b>	1037
of prioceiling attr in mutex	attr object (POSIX). /value .....	<b>pthread_mutexattr_getprioceiling()</b>	1035
initialize	Au cache library .....	<b>cacheAuLibInit()</b>	449
Alchemy	Au cache management library .....	<b>cacheAuLib</b>	38
message using MD5.	authenticate incoming RIP-2 .....	<b>ripAuthKeyInMD5()</b>	1090
message using MD5.	authenticate outgoing RIP-2 .....	<b>ripAuthKeyOut2MD5()</b>	1091
show current	authentication configuration .....	<b>ripAuthKeyShow()</b>	1091
sample	authentication hook .....	<b>ripAuthHook()</b>	1084
interface. remove	authentication hook from RIP .....	<b>ripAuthHookDelete()</b>	1087
interface. add	authentication hook to RIP .....	<b>ripAuthHookAdd()</b>	1085
add new RIP	authentication key .....	<b>ripAuthKeyAdd()</b>	1088
delete existing RIP	authentication key .....	<b>ripAuthKeyDelete()</b>	1088
find RIP	authentication key .....	<b>ripAuthKeyFind()</b>	1089
find RIP	authentication key .....	<b>ripAuthKeyFindFirst()</b>	1089
RIP-2 message. start MD5	authentication of outgoing .....	<b>ripAuthKeyOut1MD5()</b>	1090
get NFS UNIX	authentication parameters .....	<b>nfsAuthUnixGet()</b>	932
modify NFS UNIX	authentication parameters .....	<b>nfsAuthUnixPrompt()</b>	933
set NFS UNIX	authentication parameters .....	<b>nfsAuthUnixSet()</b>	933
display NFS UNIX	authentication parameters .....	<b>nfsAuthUnixShow()</b>	934
set ID number of NFS UNIX	authentication parameters .....	<b>nfsIdSet()</b>	941
library. PPP	authentication secrets .....	<b>pppSecretLib</b>	214
add secret to PPP	authentication secrets table .....	<b>pppSecretAdd()</b>	993
delete secret from PPP	authentication secrets table .....	<b>pppSecretDelete()</b>	994
display PPP	authentication secrets table .....	<b>pppSecretShow()</b>	994
enable MB86930	automatic locking of kernel/ .....	<b>cacheMb930LockAuto()</b>	459
connect routine to	auxiliary clock interrupt .....	<b>sysAuxClkConnect()</b>	1297
turn off	auxiliary clock interrupts .....	<b>sysAuxClkDisable()</b>	1297
turn on	auxiliary clock interrupts .....	<b>sysAuxClkEnable()</b>	1298
get	auxiliary clock rate .....	<b>sysAuxClkRateGet()</b>	1298
set	auxiliary clock rate .....	<b>sysAuxClkRateSet()</b>	1299
comparison routine for	AVL tree .....	<b>nextIndex()</b>	932
field. extract	backplane address from device .....	<b>bootBpAnchorExtract()</b>	432
to shared memory network	(backplane) driver. /interface .....	<b>smNetLib</b>	287
change	backspace character .....	<b>tyBackspaceSet()</b>	1407
ARM,/ get vector (trap)	base address (68K, x86, MIPS) .....	<b>intVecBaseGet()</b>	696
ARM,/ set vector (trap)	base address (68K, x86, MIPS) .....	<b>intVecBaseSet()</b>	697
specified clock for timing	base (POSIX). /timer using .....	<b>timer_create()</b>	1382
initialize	base virtual memory support .....	<b>vmBaseLibInit()</b>	1432
library.	base virtual memory support .....	<b>vmBaseLib</b>	343
compute	base-10 logarithm (ANSI) .....	<b>log10()</b>	748

	Keyword	Name	Page
	compute base-10 logarithm (ANSI).....	<b>log10f()</b>	748
	compute base-2 logarithm.....	<b>log2()</b>	747
	compute base-2 logarithm.....	<b>log2f()</b>	747
	to. compare keys based on strings they point.....	<b>hashKeyStrCmp()</b>	645
	I/O driver library. Berkeley Packet Filter (BPF).....	<b>bpfDrv</b>	35
	create Berkeley Packet Filter device.....	<b>bpfDevCreate()</b>	442
	destroy Berkeley Packet Filter device.....	<b>bpfDevDelete()</b>	442
	(ANSI). create temporary binary file (Unimplemented).....	<b>tmpfile()</b>	1391
	perform binary search (ANSI).....	<b>bsearch()</b>	443
	create and initialize binary semaphore.....	<b>semBCreate()</b>	1175
	and initialize release 4.x binary semaphore. create.....	<b>semCreate()</b>	1177
	initialize static binary semaphore.....	<b>semInit()</b>	1183
	binary semaphore library.....	<b>semBLib</b>	262
	release 4.x binary semaphore library.....	<b>semOLib</b>	271
	/and initialize shared memory binary semaphore (VxMP).....	<b>semBSmCreate()</b>	1175
	breakpoint type (MIPS). bind breakpoint handler to.....	<b>dbgBpTypeBind()</b>	513
	bind name to socket.....	<b>bind()</b>	430
	bind NPT protocol to driver.....	<b>muxTkBind()</b>	895
	port. bind socket to privileged IP.....	<b>bindresvport()</b>	431
	service and END. create binding between network.....	<b>muxBind()</b>	874
	create CBIO wrapper atop BLK_DEV device.....	<b>cbioWrapBlkDev()</b>	485
	specified physical/ show BLK_DEV structures on.....	<b>scsiBlkDevShow()</b>	1139
	size of largest available free block. find.....	<b>memPartFindMax()</b>	815
	RAM Disk Cached Block Driver.....	<b>ramDiskCbio</b>	225
	get task control block for task ID.....	<b>taskTcb()</b>	1351
	partition/ find largest free block in shared memory system.....	<b>smMemFindMax()</b>	1233
	partition. find largest free block in system memory.....	<b>memFindMax()</b>	811
	cached block I/O library.....	<b>cbioLib</b>	55
	free block of memory.....	<b>cfree()</b>	490
	reallocate block of memory (ANSI).....	<b>realloc()</b>	1065
	free block of memory (ANSI).....	<b>free()</b>	612
	set block of memory (ANSI).....	<b>memset()</b>	819
	(ANSI). search block of memory for character.....	<b>memchr()</b>	807
	partition. allocate block of memory from.....	<b>memPartAlloc()</b>	814
	memory system/ allocate block of memory from shared.....	<b>smMemMalloc()</b>	1234
	memory system/ reallocate block of memory from shared.....	<b>smMemRealloc()</b>	1235
	memory partition/ allocate block of memory from system.....	<b>malloc()</b>	802
	free block of memory in partition.....	<b>memPartFree()</b>	815
	partition. reallocate block of memory in specified.....	<b>memPartRealloc()</b>	817
	shared memory system partition block of memory (VxMP). free.....	<b>smMemFree()</b>	1233
	change state of block of virtual memory.....	<b>vmBaseStateSet()</b>	1433
	(VxVMI). change state of block of virtual memory.....	<b>vmStateSet()</b>	1444
	transfer routine. block to block (sector to sector).....	<b>cbioBlkCopy()</b>	476
	architecture-dependent page block size (VxVMI). get.....	<b>vmPageBlockSizeGet()</b>	1441
	sector) transfer routine. block to block (sector to.....	<b>cbioBlkCopy()</b>	476
	logical partition on SCSI block device. define.....	<b>scsiBlkDevCreate()</b>	1137
	read sector(s) from SCSI block device.....	<b>scsiRdSecs()</b>	1154
	write sector(s) to SCSI block device.....	<b>scsiWrtSecs()</b>	1168
	initialize file system on block device.....	<b>diskInit()</b>	544
	library. raw block device file system.....	<b>rawFsLib</b>	226

	Keyword	Name	Page
with dosFs. create	TrueFFS block device suitable for use .....	<b>tfFsDevCreate()</b>	1363
functions. associate	block device with raw volume.....	<b>rawFsDevInit()</b>	1054
/set of pending signals	blocked from delivery (POSIX).....	<b>sigpending()</b>	1221
get list of task IDs that are	blocked on semaphore. ....	<b>semInfo()</b>	1183
add to set of	blocked signals. ....	<b>sigblock()</b>	1217
lock (take) semaphore,	blocking if not available/ .....	<b>sem_wait()</b>	1195
show partition	blocks and statistics. ....	<b>memPartShow()</b>	818
show system memory partition	blocks and statistics. ....	<b>memShow()</b>	820
/shared memory system partition	blocks and statistics (VxMP). ....	<b>smMemShow()</b>	1236
read bytes or	blocks from SCSI tape device.....	<b>scsiRdTape()</b>	1154
compare two	blocks of memory (ANSI).....	<b>memcmp()</b>	807
transfer	blocks to or from memory.....	<b>cbioBlkRW()</b>	477
change	boot line. ....	<b>bootChange()</b>	432
interpret boot parameters from	boot line. ....	<b>bootStringToStruct()</b>	441
construct	boot line. ....	<b>bootStructToString()</b>	441
prompt for	boot line parameters.....	<b>bootParamsPrompt()</b>	435
display	boot line parameters. ....	<b>bootParamsShow()</b>	435
line. interpret	boot parameters from boot.....	<b>bootStringToStruct()</b>	441
retrieve	boot parameters using BOOTP.....	<b>bootpParamsGet()</b>	437
	boot ROM subroutine library. ....	<b>bootLib</b>	31
and transfer control to	boot ROMs. /network devices.....	<b>reboot()</b>	1066
configuration module for	boot ROMs. system.....	<b>bootConfig</b>	29
network with DHCP at	boot time. initialize .....	<b>dhcpcBootBind()</b>	521
device. write to	boot-image region of flash.....	<b>tfFsBootImagePut()</b>	1362
retrieve boot parameters using	BOOTP. ....	<b>bootpParamsGet()</b>	437
Bootstrap Protocol	(BOOTP) client library. ....	<b>bootpLib</b>	33
initialization.	BOOTP client library .....	<b>bootpLibInit()</b>	436
retrieve reply. send	BOOTP request message and.....	<b>bootpMsgGet()</b>	436
(VxFusion). initialize and	bootstrap current node.....	<b>distInit()</b>	549
client library.	Bootstrap Protocol (BOOTP) .....	<b>bootpLib</b>	33
DHCP	boot-time client library. ....	<b>dhcpcBootLib</b>	71
prevent strict	border gateway filtering. ....	<b>ripFilterDisable()</b>	1092
activate strict	border gateway filtering. ....	<b>ripFilterEnable()</b>	1093
socket with privileged port	bound to it. open.....	<b>rresvport()</b>	1125
allocate memory on page	boundary. ....	<b>valloc()</b>	1429
initialize	BPF driver.....	<b>bpfDrv()</b>	443
Berkeley Packet Filter	(BPF) I/O driver library. ....	<b>bpfDrv</b>	35
delete	breakpoint. ....	<b>bd()</b>	427
set hardware	breakpoint. ....	<b>bh()</b>	429
continue from	breakpoint. ....	<b>c()</b>	445
breakpoint type (MIPS). bind	breakpoint handler to.....	<b>dbgBpTypeBind()</b>	513
bind breakpoint handler to	breakpoint type (MIPS).....	<b>dbgBpTypeBind()</b>	513
set or display	breakpoints. ....	<b>b()</b>	424
delete all	breakpoints. ....	<b>bdall()</b>	428
interface. get	broadcast address for network.....	<b>ifBroadcastGet()</b>	661
interface. set	broadcast address for network.....	<b>ifBroadcastSet()</b>	661
show ports enabled for	broadcast forwarding. ....	<b>proxyPortShow()</b>	1004
particular port. disable	broadcast forwarding for .....	<b>proxyPortFwdOff()</b>	1003
particular port. enable	broadcast forwarding for .....	<b>proxyPortFwdOn()</b>	1004
change SNTP server	broadcast settings.....	<b>sntpConfigSet()</b>	1251

	Keyword	Name	Page
convert calendar time into UTC	broken-down time (ANSI).....	gmtime()	641
convert calendar time into time (ANSI), convert formatted string/ convert (ANSI), convert (POSIX), convert	broken-down time into calendar.....	localtime()	745
convert calendar time into	broken-down time into string.....	mktime()	822
convert calendar time into	broken-down time into string.....	strftime()	1275
convert calendar time into	broken-down time into string.....	asctime()	416
convert calendar time into	broken-down time into string.....	asctime_r()	416
convert calendar time into	broken-down time (ANSI).....	gmtime_r()	641
convert calendar time into	broken-down time (ANSI).....	localtime_r()	746
UNIX interface between connect	BSD 4.3 select library.....	selectLib	261
quiescent state. initialize	BSD IP protocol and MUX.....	ipProto	128
number. return	BSP serial device interrupts.....	sysSerialHwInit2()	1316
get characters from ring	BSP serial devices to.....	sysSerialHwInit()	1315
put bytes into ring	BSP version and revision.....	sysBspRev()	1299
create empty ring	buffer.....	rngBufGet()	1108
delete ring	buffer.....	rngBufPut()	1108
number of free bytes in ring	buffer.....	rngCreate()	1109
number of bytes in ring	buffer.....	rngDelete()	1109
copy data from zbuf to	buffer. determine.....	rngFreeBytes()	1110
invert order of bytes in	buffer. determine.....	rngNBytes()	1112
zero out	buffer.....	zbufExtractCopy()	1495
to client and store in	buffer.....	binvert()	431
read	buffer. /option provided.....	bzero()	444
to dot notation, store it in	buffer. /network address.....	dhcpcOptionGet()	530
copy data from mBlk to	buffer.....	fioRead()	596
<b>cacheDmaMalloc()</b> . free	buffer acquired with.....	inet_ntoa_b()	680
interrupt. clean up store	buffer after data store error.....	netMblkToBufCopy()	922
create zbuf segment from	buffer and insert into zbuf.....	cacheDmaFree()	453
write formatted string to	buffer (ANSI).....	cleanUpStoreBuffer()	493
with variable argument list to	buffer (ANSI). /formatted.....	zbufInsertBuf()	1497
copy	buffer data into zbuf.....	sprintf()	1256
make ring	buffer empty.....	vsprintf()	1446
drivers. allocate cache-safe	buffer for DMA devices and.....	zbufInsertCopy()	1498
(VxFusion). allocate telegram	buffer from pool of buffers.....	rngFlush()	1110
allocate cache-safe	buffer, if possible.....	cacheDmaMalloc()	453
test if ring	buffer is empty.....	distTBufAlloc()	556
test if ring	buffer is full (no more room).....	cacheR32kMalloc()	464
network	buffer library.....	rngIsEmpty()	1111
distributed objects telegram	buffer library (VxFusion).....	rngIsFull()	1111
	buffer manipulation library.....	netBufLib	189
disable store	buffer (MC68060 only).....	distTBufLib	85
enable store	buffer (MC68060 only).....	bLib	28
dynamic ring	buffer (rBuff) library.....	cacheStoreBufDisable()	470
CL_POOL_ID for specified	buffer size. return.....	cacheStoreBufEnable()	470
remote file device with fixed	buffer size. create.....	rBuffLib	230
ring	buffer subroutine library.....	netCIPoolIdGet()	910
compare one	buffer to another.....	netDevCreate2()	912
copy one	buffer to another.....	rngLib	237
a time. copy one	buffer to another one byte at.....	bcmp()	425
		bcopy()	425
		bcopyBytes()	426

	Keyword	Name	Page
word at a time. copy one	buffer to another one long .....	<b>bcopyLongs()</b>	426
a time. copy one	buffer to another one word at .....	<b>bcopyWords()</b>	427
(VxFusion). return telegram	buffer to pool of buffers .....	<b>distTBufFree()</b>	557
return ID of WindView event	buffer (WindView).....	<b>wvEvtBufferGet()</b>	1470
start logging events to	buffer (WindView).....	<b>wvEvtLogStart()</b>	1472
stop logging events to	buffer (WindView).....	<b>wvEvtLogStop()</b>	1473
character. fill	buffer with specified.....	<b>bfill()</b>	428
character one byte at a/ fill	buffer with specified.....	<b>bfillBytes()</b>	429
put byte ahead in ring	buffer without moving ring/ .....	<b>rngPutAhead()</b>	1113
or standard error. set line	buffering for standard output.....	<b>setlinebuf()</b>	1202
specify	buffering for stream.....	<b>setbuffer()</b>	1200
specify	buffering for stream (ANSI).....	<b>setbuf()</b>	1200
specify	buffering for stream (ANSI).....	<b>setvbuf()</b>	1211
show state of Pty	Buffers.....	<b>ptyShow()</b>	1045
swap	buffers.....	<b>bswap()</b>	444
TLBs (Translation Lookaside	Buffers). flush .....	<b>pentiumTlbFlush()</b>	973
necessarily/ swap bytes with	buffers that are not.....	<b>uswab()</b>	1426
flush processor write	buffers to memory.....	<b>cachePipeFlush()</b>	460
telegram buffer from pool of	buffers (VxFusion). allocate.....	<b>distTBufAlloc()</b>	556
telegram buffer to pool of	buffers (VxFusion). return .....	<b>distTBufFree()</b>	557
pulse reset signal on SCSI	bus.....	<b>scsiBusReset()</b>	1139
test and set location across	bus.....	<b>sysBusTas()</b>	1301
convert local address to	bus address.....	<b>sysLocalToBusAdrs()</b>	1306
convert	bus address to local address.....	<b>sysBusToLocalAdrs()</b>	1301
probe address for	bus error.....	<b>vxMemProbe()</b>	1454
acknowledge	bus interrupt.....	<b>sysBusIntAck()</b>	1300
generate	bus interrupt.....	<b>sysBusIntGen()</b>	1300
disable	bus interrupt level.....	<b>sysIntDisable()</b>	1305
enable	bus interrupt level.....	<b>sysIntEnable()</b>	1305
assert RST line on SCSI	bus (Western Digital WD33C93/ .....	<b>sysScsiBusReset()</b>	1312
interface. remove table	bypass hook from RIP .....	<b>ripLeakHookDelete()</b>	1097
tables. add hook to	bypass RIP and kernel routing.....	<b>ripLeakHookAdd()</b>	1096
advance ring pointer by n	bytes.....	<b>rngMoveAhead()</b>	1112
swap	bytes.....	<b>swab()</b>	1287
read	bytes from file or device.....	<b>read()</b>	1064
delete	bytes from zbuf.....	<b>zbufCut()</b>	1492
invert order of	bytes in buffer.....	<b>binvert()</b>	431
determine number of free	bytes in ring buffer.....	<b>rngFreeBytes()</b>	1110
determine number of	bytes in ring buffer.....	<b>rngNBytes()</b>	1112
put	bytes into ring buffer.....	<b>rngBufPut()</b>	1108
determine length in	bytes of zbuf.....	<b>zbufLength()</b>	1499
device. read	bytes or blocks from SCSI tape .....	<b>scsiRdTape()</b>	1154
write	bytes to file.....	<b>write()</b>	1468
transfer	bytes to or from memory.....	<b>cbioBytesRW()</b>	477
not necessarily aligned. swap	bytes with buffers that are .....	<b>uswab()</b>	1426
to user-defined function	(C++). set new_handler.....	<b>set_new_handler()</b>	1199
to user-defined function	(C++). set terminate.....	<b>set_terminate()</b>	1199
allocation failure handler	(C++). call.....	<b>cplusCallNewHandler()</b>	502
call static constructors	(C++). .....	<b>cplusCtors()</b>	503
all linked static constructors	(C++). call.....	<b>cplusCtorsLink()</b>	504

	Keyword	Name	Page
change C++ demangling mode	(C++). .....	<code>cplusDemanglerSet()</code>	504
change C++ demangling style	(C++). .....	<code>cplusDemanglerStyleSet()</code>	505
call static destructors	(C++). .....	<code>cplusDtors()</code>	505
all linked static destructors	(C++). call .....	<code>cplusDtorsLink()</code>	506
initialize C++ library	(C++). .....	<code>cplusLibInit()</code>	507
constructor calling strategy	(C++). change C++ static .....	<code>cplusXtorSet()</code>	507
basic run-time support for	C++. .....	<code>cplusLib</code>	62
for memory deallocation	(C++). /run-time support .....	<code>operator delete()</code>	946
support for operator new	(C++). default run-time .....	<code>operator new()</code>	947
for operator new (nothrow)	(C++). /run-time support .....	<code>operator new()</code>	947
operator new with placement	(C++). run-time support for .....	<code>operator new()</code>	948
change	C++ demangling mode (C++). .....	<code>cplusDemanglerSet()</code>	504
change	C++ demangling style (C++). .....	<code>cplusDemanglerStyleSet()</code>	505
high-level math functions.	C interface library to .....	<code>mathALib</code>	153
initialize	C++ library (C++). .....	<code>cplusLibInit()</code>	507
/interrupt handler for	C routine (68K, x86, MIPS, / .....	<code>intHandlerCreate()</code>	688
exception vector/ connect	C routine to asynchronous .....	<code>excIntConnect()</code>	580
exception vector/ connect	C routine to critical. ....	<code>excCrtConnect()</code>	578
interrupt vector/ connect	C routine to critical. ....	<code>excIntCrtConnect()</code>	581
(PowerPC). connect	C routine to exception vector .....	<code>excConnect()</code>	577
interrupt. connect	C routine to hardware. ....	<code>intConnect()</code>	683
interrupt handler for	C routine (x86). construct .....	<code>intHandlerCreateI86()</code>	689
strategy (C++). change	C++ static constructor calling .....	<code>cplusXtorSet()</code>	507
clear all or some entries from	cache. ....	<code>cacheClear()</code>	449
clear line from CY7C604	cache. ....	<code>cacheCy604ClearLine()</code>	450
clear page from CY7C604	cache. ....	<code>cacheCy604ClearPage()</code>	450
clear region from CY7C604	cache. ....	<code>cacheCy604ClearRegion()</code>	451
clear segment from CY7C604	cache. ....	<code>cacheCy604ClearSegment()</code>	451
disable specified	cache. ....	<code>cacheDisable()</code>	452
enable specified	cache. ....	<code>cacheEnable()</code>	456
flush all or some of specified	cache. ....	<code>cacheFlush()</code>	456
all or some of specified	cache. invalidate .....	<code>cacheInvalidate()</code>	457
lock all or part of specified	cache. ....	<code>cacheLock()</code>	458
clear line from MB86930	cache. ....	<code>cacheMb930ClearLine()</code>	458
specific context from Sun-4	cache. clear .....	<code>cacheSun4ClearContext()</code>	470
clear line from Sun-4	cache. ....	<code>cacheSun4ClearLine()</code>	471
clear page from Sun-4	cache. ....	<code>cacheSun4ClearPage()</code>	471
clear segment from Sun-4	cache. ....	<code>cacheSun4ClearSegment()</code>	472
all or part of specified	cache. unlock .....	<code>cacheUnlock()</code>	475
create disk	cache. ....	<code>dcacheDevCreate()</code>	514
re-enable disk	cache. ....	<code>dcacheDevEnable()</code>	516
print information about disk	cache. ....	<code>dcacheShow()</code>	519
clear entry from	cache (68K, x86). .....	<code>cacheArchClearEntry()</code>	446
set new size to disk	cache device. ....	<code>dcacheDevMemResize()</code>	516
disk	cache driver. ....	<code>dcacheCbio</code>	67
flush data	cache for drivers. ....	<code>cacheDrvFlush()</code>	454
invalidate data	cache for drivers. ....	<code>cacheDrvInvalidate()</code>	454
disable disk	cache for this device. ....	<code>dcacheDevDisable()</code>	515
initialize 4kc	cache library. ....	<code>cache4kcLibInit()</code>	445
initialize	cache library. ....	<code>cacheArchLibInit()</code>	447

	Keyword	Name	Page
	initialize Au	cache library..... <b>cacheAuLibInit()</b>	449
initialize Cypress CY7C604	cache library.....	<b>cacheCy604LibInit()</b>	452
initialize Fujitsu MB86930	cache library.....	<b>cacheMb930LibInit()</b>	459
initialize R3000	cache library.....	<b>cacheR3kLibInit()</b>	460
initialize R4000	cache library.....	<b>cacheR4kLibInit()</b>	461
initialize R5000	cache library.....	<b>cacheR5kLibInit()</b>	461
initialize R7000	cache library.....	<b>cacheR7kLibInit()</b>	462
initialize R10000	cache library.....	<b>cacheR10kLibInit()</b>	463
initialize RC32364	cache library.....	<b>cacheR32kLibInit()</b>	463
initialize R33000	cache library.....	<b>cacheR33kLibInit()</b>	464
initialize R333x0	cache library.....	<b>cacheR333x0LibInit()</b>	465
initialize SH7040	cache library.....	<b>cacheSh7040LibInit()</b>	465
initialize SH7604/SH7615	cache library.....	<b>cacheSh7604LibInit()</b>	466
initialize SH7622	cache library.....	<b>cacheSh7622LibInit()</b>	466
initialize SH7700	cache library.....	<b>cacheSh7700LibInit()</b>	467
initialize SH7729	cache library.....	<b>cacheSh7729LibInit()</b>	468
initialize SH7750	cache library.....	<b>cacheSh7750LibInit()</b>	469
initialize Sun-4	cache library.....	<b>cacheSun4LibInit()</b>	472
initialize TI TMS390	cache library.....	<b>cacheTiTms390LibInit()</b>	473
initialize Tx49	cache library.....	<b>cacheTx49LibInit()</b>	475
architecture. initialize	cache library for processor.....	<b>cacheLibInit()</b>	457
MIPS 4kc	cache management library.....	<b>cache4kcLib</b>	37
architecture-specific	cache management library.....	<b>cacheArchLib</b>	37
Alchemy Au	cache management library.....	<b>cacheAuLib</b>	38
MIPS R3000	cache management library.....	<b>cacheR3kLib</b>	47
MIPS R4000	cache management library.....	<b>cacheR4kLib</b>	47
MIPS R5000	cache management library.....	<b>cacheR5kLib</b>	48
MIPS R7000	cache management library.....	<b>cacheR7kLib</b>	48
MIPS R10000	cache management library.....	<b>cacheR10kLib</b>	49
MIPS RC32364	cache management library.....	<b>cacheR32kLib</b>	49
MIPS R333x0	cache management library.....	<b>cacheR333x0Lib</b>	50
MIPS R33000	cache management library.....	<b>cacheR33kLib</b>	50
Hitachi SH7040	cache management library.....	<b>cacheSh7040Lib</b>	51
Hitachi SH7604/SH7615	cache management library.....	<b>cacheSh7604Lib</b>	51
SH7622	cache management library.....	<b>cacheSh7622Lib</b>	52
Hitachi SH7700	cache management library.....	<b>cacheSh7700Lib</b>	52
Hitachi SH7729	cache management library.....	<b>cacheSh7729Lib</b>	53
Hitachi SH7750	cache management library.....	<b>cacheSh7750Lib</b>	53
Sun-4	cache management library.....	<b>cacheSun4Lib</b>	54
Toshiba Tx49	cache management library.....	<b>cacheTx49Lib</b>	54
modify tunable disk	cache parameters.....	<b>dcacheDevTune()</b>	517
RAM Disk	Cached Block Driver.....	<b>ramDiskCbio</b>	225
	cached block I/O library.....	<b>cbioLib</b>	55
free buffer acquired with	<b>cacheDmaMalloc()</b> .....	<b>cacheDmaFree()</b>	453
translate virtual address for	cacheLib.....	<b>cacheTiTms390VirtToPhys()</b>	474
instruction and data	caches. synchronize.....	<b>cacheTextUpdate()</b>	473
synchronize	caches for data coherency.....	<b>scsiCacheSynchronize()</b>	1141
SCSI that hardware snooping of	caches is disabled. inform.....	<b>scsiCacheSnoopDisable()</b>	1140
SCSI that hardware snooping of	caches is enabled. inform.....	<b>scsiCacheSnoopEnable()</b>	1140

	Keyword	Name	Page
devices and drivers. allocate possible. allocate	cache-safe buffer for DMA	<b>cacheDmaMalloc()</b>	453
determine current	cache-safe buffer, if	<b>cacheR32kMalloc()</b>	464
convert broken-down time into time (ANSI). convert	calendar time (ANSI)	<b>time()</b>	1380
time (ANSI). convert	calendar time (ANSI)	<b>mktime()</b>	822
time (POSIX). convert	calendar time into broken-down	<b>localtime()</b>	745
time (POSIX). convert	calendar time into broken-down	<b>gmtime_r()</b>	641
broken-down time/ convert	calendar time into broken-down	<b>localtime_r()</b>	746
compute difference between two thread (POSIX). create	calendar time into UTC	<b>gmtime()</b>	641
thread (POSIX). set	calendar times (ANSI)	<b>difftime()</b>	542
thread (POSIX). set	cancellation point in calling	<b>pthread_testcancel()</b>	1043
set	cancellation state for calling	<b>pthread_setcancelstate()</b>	1039
return mode setting for	cancellation type for calling	<b>pthread_setcanceltype()</b>	1040
set mode for	case sensitivity of volume	<b>dosSetVolCaseSens()</b>	563
determine ready status of	CBIO device.	<b>cbioModeGet()</b>	481
change in ready status of	CBIO device.	<b>cbioModeSet()</b>	481
print information about	CBIO device.	<b>cbioRdyChgdGet()</b>	482
initialize	CBIO device. force	<b>cbioRdyChgdSet()</b>	483
in CBIO_PARAMS structure with	CBIO device.	<b>cbioShow()</b>	484
obtain	CBIO device (Generic)	<b>cbioDevCreate()</b>	478
release	CBIO device parameters. fill	<b>cbioParamsGet()</b>	482
Initialize	CBIO device semaphore.	<b>cbioLock()</b>	480
device. create	CBIO device semaphore.	<b>cbioUnlock()</b>	484
verify	CBIO Library.	<b>cbioLibInit()</b>	480
CBIO device/ fill in	CBIO wrapper atop BLK_DEV	<b>cbioWrapBlkDev()</b>	485
primitive.	CBIO_DEV_ID	<b>cbioDevVerify()</b>	478
system from ATA hard disk or library. ISO 9660	CBIO_PARAMS structure with	<b>cbioParamsGet()</b>	482
initialize	C-callable atomic test-and-set	<b>vxTas()</b>	1459
create	CDROM. mount DOS file	<b>usrAtaConfig()</b>	1419
duplicate mBlk	CD-ROM read-only file system	<b>cdromFsLib</b>	59
constructs. free	cdromFsLib	<b>cdromFsInit()</b>	488
device associated with serial VxWorks device for serial	cdromFsLib device.	<b>cdromFsDevCreate()</b>	488
device access to serial	chain.	<b>netMblkChainDup()</b>	916
change abort	chain of mBlk-clBlk-cluster	<b>netMblkCIChainFree()</b>	917
change backspace	chains two triggers.	<b>trgChainSet()</b>	1395
change line-delete	channel. get SIO_CHAN	<b>sysSerialChanGet()</b>	1315
change end-of-file	channel. create	<b>ttyDevCreate()</b>	1405
change trap-to-monitor	channels. provide terminal	<b>ttyDrv</b>	326
fill buffer with specified	character.	<b>tyAbortSet()</b>	1406
is printing, non-white-space	character.	<b>tyBackspaceSet()</b>	1407
is printable, including space	character.	<b>tyDeleteLineSet()</b>	1407
character is white-space	character.	<b>tyEOFSet()</b>	1409
search block of memory for	character.	<b>tyMonitorTrapSet()</b>	1411
stream (ANSI). push	character.	<b>bfill()</b>	428
/string length up to first	character (ANSI). /character	<b>isgraph()</b>	722
first occurrence in string of	character (ANSI). /character	<b>isprint()</b>	723
	character (ANSI). /whether	<b>isspace()</b>	724
	character (ANSI).	<b>memchr()</b>	807
	character back into input	<b>ungetc()</b>	1413
	character from given set/	<b>strcspn()</b>	1273
	character from given set/ find	<b>strpbrk()</b>	1279

	Keyword	Name	Page
stream (ANSI). return next	character from standard input .....	<b>getchar()</b>	635
return next	character from stream (ANSI).....	<b>fgetc()</b>	590
return next	character from stream (ANSI).....	<b>getc()</b>	635
find last occurrence of	character in string.....	<b>rindex()</b>	1083
find first occurrence of	character in string.....	<b>index()</b>	674
find first occurrence of	character in string (ANSI).....	<b>strchr()</b>	1271
find last occurrence of	character in string (ANSI).....	<b>strrchr()</b>	1279
(ANSI). test whether	character is alphanumeric.....	<b>isalnum()</b>	719
(ANSI). test whether	character is control character .....	<b>iscntrl()</b>	721
(ANSI). test whether	character is decimal digit.....	<b>isdigit()</b>	721
(ANSI). test whether	character is hexadecimal digit.....	<b>isxdigit()</b>	725
test whether	character is letter (ANSI).....	<b>isalpha()</b>	720
(ANSI). test whether	character is lower-case letter .....	<b>islower()</b>	722
including space/ test whether	character is printable,.....	<b>isprint()</b>	723
non-white-space/ test whether	character is printing.....	<b>isgraph()</b>	722
(ANSI). test whether	character is punctuation.....	<b>ispunct()</b>	723
(ANSI). test whether	character is upper-case letter.....	<b>isupper()</b>	724
character/ test whether	character is white-space .....	<b>isspace()</b>	724
/string length up to first	character not in given set/.....	<b>strspn()</b>	1280
fill buffer with specified	character one byte at a time.....	<b>bfillBytes()</b>	429
character/ convert wide	character to multibyte .....	<b>wctomb()</b>	1461
stream (ANSI). write	character to standard output.....	<b>putc()</b>	1046
write	character to stream (ANSI).....	<b>putc()</b>	1045
write	character to stream (ANSI).....	<b>fputc()</b>	610
convert multibyte	character to wide character/ .....	<b>mbtowc()</b>	805
/wide character to multibyte	character (Unimplemented)/ .....	<b>wctomb()</b>	1461
calculate length of multibyte	character (Unimplemented)/ .....	<b>mblen()</b>	804
/multibyte character to wide	character (Unimplemented)/ .....	<b>mbtowc()</b>	805
(ANSI). read and convert	characters from ASCII string.....	<b>sscanf()</b>	1263
another (ANSI). concatenate	characters from one string to.....	<b>strncat()</b>	1277
another (ANSI). copy	characters from one string to.....	<b>strncpy()</b>	1278
get	characters from ring buffer.....	<b>rngBufGet()</b>	1108
stream/ read and convert	characters from standard input .....	<b>scanf()</b>	1130
stream (ANSI). read	characters from standard input .....	<b>gets()</b>	638
read specified number of	characters from stream (ANSI).....	<b>fgets()</b>	591
read and convert	characters from stream (ANSI).....	<b>fscanf()</b>	614
(ANSI). transform up to n	characters of s2 into s1 .....	<b>strxfrm()</b>	1287
(ANSI). compare first n	characters of two strings.....	<b>strncmp()</b>	1278
convert series of wide	char's to multibyte char's/ .....	<b>wcstombs()</b>	1461
convert series of multibyte	char's to wide char's/ .....	<b>mbstowcs()</b>	804
/of wide char's to multibyte	char's (Unimplemented) (ANSI).....	<b>wcstombs()</b>	1461
/of multibyte char's to wide	char's (Unimplemented) (ANSI).....	<b>mbstowcs()</b>	804
verify	checksums on all modules.....	<b>moduleCheck()</b>	827
being logged/ clear	class of events from those .....	<b>wvEvtClassClear()</b>	1470
(WindView). set	class of events to log .....	<b>wvEvtClassSet()</b>	1471
get current set of	classes being logged/ .....	<b>wvEvtClassGet()</b>	1471
logged (WindView). clear all	classes of events from those .....	<b>wvEvtClassClearAll()</b>	1470
get	clBlk.....	<b>netCIBlkGet()</b>	908
join cluster to	clBlk structure.....	<b>netCIBlkJoin()</b>	908
specified mBlk. get	clBlk-cluster and join it to .....	<b>netMblkClGet()</b>	918

	Keyword	Name	Page
	join mBlk to	clBlk-cluster construct.....	netMblkCJoin() 919
	to memory pool. free	clBlk-cluster construct back.....	netCIBlkFree() 907
	register proxy	client.....	proxyReg() 1005
	unregister proxy	client.....	proxyUnreg() 1005
	retrieve option provided to	client and store in buffer.....	dhcpcOptionGet() 530
	Protocol (DHCP) run-time	client API. /Configuration.....	dhcpcLib 73
	routines. DHCP run-time	client information display.....	dhcpcShow 75
	library. DHCP	client interface shared code.....	dhcpcCommonLib 72
	Resolution Protocol (ARP)	client library. proxy Address.....	proxyLib 216
	Network Time Protocol (SNTP)	client library. Simple.....	sntpLib 291
	File Transfer Protocol (TFTP)	client library. Trivial.....	tftpLib 319
	Bootstrap Protocol (BOOTP)	client library.....	bootpLib 33
	disable DHCP	client library.....	dhcpcShutdown() 536
	DHCP boot-time	client library.....	dhcpcBootLib 71
	BOOTP	client library initialization.....	bootpLibInit() 436
	DHCP	client library initialization.....	dhcpcLibInit() 528
	add option to	client messages.....	dhcpcOptionAdd() 529
	structures. set up DHCP	client parameters and data.....	dhcpcBootInit() 522
	routine to access reference	clock. assign.....	sntpsClockSet() 1250
	set alarm	clock for delivery of signal.....	alarm() 411
	allocate timer using specified	clock for timing base (POSIX).....	timer_create() 1382
	connect routine to auxiliary	clock interrupt.....	sysAuxClkConnect() 1297
	connect routine to system	clock interrupt.....	sysClkConnect() 1302
	user-defined system	clock interrupt routine.....	usrClock() 1420
	turn off auxiliary	clock interrupts.....	sysAuxClkDisable() 1297
	turn on auxiliary	clock interrupts.....	sysAuxClkEnable() 1298
	turn off system	clock interrupts.....	sysClkDisable() 1302
	turn on system	clock interrupts.....	sysClkEnable() 1303
	clock library (POSIX).	clock library (POSIX).....	clockLib 61
	get current time of	clock (POSIX).....	clock_gettime() 495
	get auxiliary	clock rate.....	sysAuxClkRateGet() 1298
	set auxiliary	clock rate.....	sysAuxClkRateSet() 1299
	get system	clock rate.....	sysClkRateGet() 1303
	set system	clock rate.....	sysClkRateSet() 1304
	set	clock resolution.....	clock_setres() 495
	get	clock resolution (POSIX).....	clock_getres() 494
	announce	clock tick support library.....	tickLib 321
	(POSIX). set	clock tick to kernel.....	tickAnnounce() 1379
		clock to specified time.....	clock_settime() 496
		close active telnet session.....	telnetdExit() 1359
		close directory (POSIX).....	closedir() 497
	(WindView).	close event-destination file.....	fileUploadPathClose() 592
		close file.....	close() 496
		close message queue (POSIX).....	mq_close() 838
		close named semaphore (POSIX).....	sem_close() 1188
	(Windview).	close socket upload path.....	sockUploadPathClose() 1253
		close stream (ANSI).....	fclose() 587
	(Windview).	close TSFS-socket upload path.....	tsfsUploadPathClose() 1402
	set TCP connection to	closed state.....	m2TcpConnEntrySet() 798
	buffer size. return	CL_POOL_ID for specified.....	netCIPoolIdGet() 910

	Keyword	Name	Page
	free cluster back to memory pool.....	<b>netCIFree()</b>	909
	pool. get cluster from specified cluster .....	<b>netClusterGet()</b>	911
	get cluster from specified cluster pool.....	<b>netClusterGet()</b>	911
	join cluster to cBlk structure. ....	<b>netCIBlkJoin()</b>	908
	and get complete RFC reply code. send FTP command.....	<b>ftpCommandEnhanced()</b>	622
	inflate compressed code. ....	<b>inflate()</b>	682
	DHCP client interface shared code library.....	<b>dhcpcCommonLib</b>	72
	functions. inflate code using public domain zlib .....	<b>inflateLib</b>	123
	synchronize caches for data coherency.....	<b>scsiCacheSynchronize()</b>	1141
	(SCSI-2). SCSI library common commands for all devices .....	<b>scsiCommonLib</b>	256
	devices/ SCSI library common commands for all.....	<b>scsiCommonLib</b>	256
	get common values. ....	<b>m2IfCommonValsGet()</b>	774
	two strings (ANSI). compare first n characters of .....	<b>strncmp()</b>	1278
	identifiers. compare keys as 32 bit.....	<b>hashKeyCmp()</b>	645
	they point to. compare keys based on strings .....	<b>hashKeyStrCmp()</b>	645
	compare one buffer to another. ....	<b>bcmp()</b>	425
	compare thread IDs (POSIX). ....	<b>pthread_equal()</b>	1026
	(ANSI). compare two blocks of memory .....	<b>memcmp()</b>	807
	appropriate to LC_COLLATE/ compare two strings as .....	<b>strcoll()</b>	1272
	lexicographically (ANSI). compare two strings .....	<b>strcmp()</b>	1272
	instruction to/ execute atomic compare-and-exchange .....	<b>pentiumBtc()</b>	951
	instruction to/ execute atomic compare-and-exchange .....	<b>pentiumBts()</b>	951
	UNIX tar compatible library. ....	<b>tarLib</b>	306
	format MS-DOS compatible volume. ....	<b>dosFsVolFormat()</b>	562
	low level I/O access to flash components.....	<b>tffsRawio()</b>	1366
	lconv (ANSI). set components of object with type.....	<b>localeconv()</b>	742
	packet. compress DNS name in DNS .....	<b>resolvDNComp()</b>	1074
	inflate compressed code.....	<b>inflate()</b>	682
	packet. expand DNS compressed name from DNS .....	<b>resolvDNExpand()</b>	1074
	one string to another (ANSI). concatenate characters from.....	<b>strncat()</b>	1277
	another (ANSI). concatenate one string to .....	<b>strcat()</b>	1271
	concatenate two lists.....	<b>lstConcat()</b>	762
	(POSIX). initialize condition attribute object.....	<b>pthread_condattr_init()</b>	1024
	(POSIX). destroy condition attributes object .....	<b>pthread_condattr_destroy()</b>	1024
	unblock all threads waiting on condition (POSIX). ....	<b>pthread_cond_broadcast()</b>	1020
	unblock thread waiting on condition (POSIX). ....	<b>pthread_cond_signal()</b>	1022
	destroy condition variable (POSIX).....	<b>pthread_cond_destroy()</b>	1020
	initialize condition variable (POSIX).....	<b>pthread_cond_init()</b>	1021
	wait for condition variable (POSIX).....	<b>pthread_cond_wait()</b>	1023
	timeout (POSIX). wait for condition variable with.....	<b>pthread_cond_timedwait()</b>	1022
	show current authentication configuration. ....	<b>ripAuthKeyShow()</b>	1091
	system SCSI configuration. ....	<b>sysScsiConfig()</b>	1313
	changes. alter RIP configuration after interface .....	<b>ripIfReset()</b>	1095
	display dosFs volume configuration data.....	<b>dosFsShow()</b>	561
	VxWorks. TrueFFS configuration file for.....	<b>tffsConfig</b>	315
	show volume configuration information. ....	<b>cdromFsVolConfigShow()</b>	489
	requested NFS device. read configuration information from.....	<b>nfsDevInfoGet()</b>	934
	user-defined system configuration library. ....	<b>usrConfig</b>	336
	ROMs. system configuration module for boot.....	<b>bootConfig</b>	29
	registered with MUX. display configuration of devices.....	<b>muxShow()</b>	893

	Keyword	Name	Page
	add routine to handle configuration parameters.....	<b>dhcpcEventHookAdd()</b>	525
	retrieve current configuration parameters.....	<b>dhcpcParamsGet()</b>	533
	handler. remove configuration parameters.....	<b>dhcpcEventHookDelete()</b>	526
	DHCP. obtain set of network configuration parameters with.....	<b>dhcpcBind()</b>	520
	DHCP. obtain additional configuration parameters with.....	<b>dhcpcBootInformGet()</b>	521
	DHCP. obtain additional configuration parameters with.....	<b>dhcpcInformGet()</b>	526
	run-time client/ Dynamic Host Configuration Protocol (DHCP).....	<b>dhcpcLib</b>	73
	server library. Dynamic Host Configuration Protocol (DHCP).....	<b>dhcpsLib</b>	76
	connected to SCSI controller. configure all devices .....	<b>scsiAutoConfig()</b>	1137
	unnumbered. configure interface to be .....	<b>ifUnnumberedSet()</b>	672
	configure SCSI peripherals. ....	<b>usrScsiConfig()</b>	1425
	interrupts. connect BSP serial device .....	<b>sysSerialHwInit2()</b>	1316
	asynchronous exception vector/ connect C routine to .....	<b>excIntConnect()</b>	580
	exception vector (PowerPC/ connect C routine to critical .....	<b>excCrtConnect()</b>	578
	interrupt vector (PowerPC/ connect C routine to critical .....	<b>excIntCrtConnect()</b>	581
	vector (PowerPC). connect C routine to exception.....	<b>excConnect()</b>	577
	interrupt. connect C routine to hardware.....	<b>intConnect()</b>	683
	clock interrupt. connect routine to auxiliary .....	<b>sysAuxClkConnect()</b>	1297
	interrupt. connect routine to mailbox .....	<b>sysMailboxConnect()</b>	1306
	clock interrupt. connect routine to system .....	<b>sysClkConnect()</b>	1302
	signal. connect user routine to timer.....	<b>timer_connect()</b>	1381
	get name of connected peer. ....	<b>getpeername()</b>	637
	configure all devices connected to SCSI controller. ....	<b>scsiAutoConfig()</b>	1137
	shut down network connection. ....	<b>shutdown()</b>	1216
	get completed FTP data connection. ....	<b>ftpDataConnGet()</b>	623
	accept connection from socket.....	<b>accept()</b>	404
	target host connection library using TSFS.....	<b>wvTsfsUploadPathLib</b>	358
	get MIB-II TCP connection table entry.....	<b>m2TcpConnEntryGet()</b>	797
	set TCP connection to closed state.....	<b>m2TcpConnEntrySet()</b>	798
	specified host. get control connection to FTP server on .....	<b>ftpHookup()</b>	626
	initiate connection to socket.....	<b>connect()</b>	497
	initialize FTP data connection using PASV mode. .	<b>ftpDataConnInitPassiveMode()</b>	624
	initialize FTP data connection using PORT mode.....	<b>ftpDataConnInit()</b>	623
	duration. attempt socket connection within specified .....	<b>connectWithTimeout()</b>	498
	protocol/ display all active connections for Internet.....	<b>inetstatShow()</b>	681
	enable connections to socket. ....	<b>listen()</b>	736
	file system. perform consistency checking on MS-DOS.....	<b>chkdsk()</b>	492
	(C++). change C++ static constructor calling strategy.....	<b>cplusXtorSet()</b>	507
	call static constructors (C++).....	<b>cplusCtors()</b>	503
	call all linked static constructors (C++).....	<b>cplusCtorsLink()</b>	504
	attributes (POSIX). set contention scope for thread .....	<b>pthread_attr_setscope()</b>	1017
	attributes (POSIX). get contention scope from thread .....	<b>pthread_attr_getscope()</b>	1011
	floating-point coprocessor context. restore.....	<b>fppRestore()</b>	601
	floating-point coprocessor context. save.....	<b>fppSave()</b>	603
	state is in interrupt or task context. determine if current .....	<b>intContext()</b>	686
	clear specific context from Sun-4 cache. ....	<b>cacheSun4ClearContext()</b>	470
	WDB agent context management library. ....	<b>wdbLib</b>	348
	create new virtual memory context (VxVMI). ....	<b>vmContextCreate()</b>	1434
	delete virtual memory context (VxVMI). ....	<b>vmContextDelete()</b>	1435
	display translation table for context (VxVMI). ....	<b>vmContextShow()</b>	1435

	Keyword	Name	Page
get current virtual memory	context (VxVMI).....	<b>vmCurrentGet()</b>	1436
set current virtual memory	context (VxVMI).....	<b>vmCurrentSet()</b>	1436
	continue from breakpoint.....	<b>c()</b>	445
parameters. initiate or	continue negotiating transfer.....	<b>scsiSyncXferNegotiate()</b>	1162
parameters. initiate or	continue negotiating wide.....	<b>scsiWideXferNegotiate()</b>	1167
subroutine returns.	continue until current.....	<b>cret()</b>	509
get task	control block for task ID.....	<b>taskTcb()</b>	1351
server on specified host. get	control connection to FTP.....	<b>ftpHookup()</b>	626
ICMP router discovery	control function. implement.....	<b>rdCtl()</b>	1059
perform device-specific I/O	control function.....	<b>scsiIoctl()</b>	1145
perform I/O	control function.....	<b>ioctl()</b>	704
sequential access/ perform I/O	control function for.....	<b>scsiSeqIoctl()</b>	1159
perform distributed objects	control function (VxFusion).....	<b>distCtl()</b>	544
to device. send	control information to MUX or.....	<b>muxIoctl()</b>	880
trigger events	control library.....	<b>trgLib</b>	324
/objects initialization and	control library (VxFusion).....	<b>distLib</b>	83
event logging	control library (WindView).....	<b>wvLib</b>	351
attempts to take spin-lock/	control logging of failed.....	<b>smObjTimeoutLogEnable()</b>	1248
get content of	Control Register 0 (x86).....	<b>vxCr0Get()</b>	1449
set value to	Control Register 0 (x86).....	<b>vxCr0Set()</b>	1450
get content of	Control Register 2 (x86).....	<b>vxCr2Get()</b>	1447
set value to	Control Register 2 (x86).....	<b>vxCr2Set()</b>	1447
get content of	Control Register 3 (x86).....	<b>vxCr3Get()</b>	1448
set value to	Control Register 3 (x86).....	<b>vxCr3Set()</b>	1448
get content of	Control Register 4 (x86).....	<b>vxCr4Get()</b>	1448
set value to	Control Register 4 (x86).....	<b>vxCr4Set()</b>	1449
handle device	control requests.....	<b>tyIoctl()</b>	1409
get parameters which	control resolver library.....	<b>resolveParamsGet()</b>	1078
set parameters which	control resolver library.....	<b>resolveParamsSet()</b>	1079
network devices and transfer	control to boot ROMs. reset.....	<b>reboot()</b>	1066
transfer	control to ROM monitor.....	<b>sysToMonitor()</b>	1317
test whether character is	control character (ANSI).....	<b>iscntrl()</b>	721
all devices connected to SCSI	controller. configure.....	<b>scsiAutoConfig()</b>	1137
devices attached to SCSI	controller. list physical.....	<b>scsiShow()</b>	1160
notify SCSI manager of SCSI	(controller) event.....	<b>scsiMgrEventNotify()</b>	1147
SCSI thread-level	controller library (SCSI-2).....	<b>scsiCtrlLib</b>	256
send event to SCSI	controller state machine.....	<b>scsiMgrCtrlEvent()</b>	1146
handle	controller-bus reset event.....	<b>scsiMgrBusReset()</b>	1146
calendar time (ANSI).	convert broken-down time into.....	<b>mktime()</b>	822
formatted string (ANSI).	convert broken-down time into.....	<b>strftime()</b>	1275
string (ANSI).	convert broken-down time into.....	<b>asctime()</b>	416
string (POSIX).	convert broken-down time into.....	<b>asctime_r()</b>	416
address.	convert bus address to local.....	<b>sysBusToLocalAdrs()</b>	1301
broken-down time (ANSI).	convert calendar time into.....	<b>localtime()</b>	745
broken-down time (POSIX).	convert calendar time into.....	<b>gmtime_r()</b>	641
broken-down time (POSIX).	convert calendar time into.....	<b>localtime_r()</b>	746
broken-down time (ANSI).	convert calendar time into UTC.....	<b>gmtime()</b>	641
string (ANSI). read and	convert characters from ASCII.....	<b>sscanf()</b>	1263
standard input/ read and	convert characters from.....	<b>scanf()</b>	1130
(ANSI). read and	convert characters from stream.....	<b>fscanf()</b>	614

	Keyword	Name	Page
volume descriptor pointer.	convert device name into DOS.....	<b>dosFsVolDescGet()</b>	561
address to long integer.	convert dot notation Internet.....	<b>inet_addr()</b>	675
to integer.	convert double-precision value.....	<b>irint()</b>	717
	convert format string .....	<b>fiFormatV()</b>	594
local address (VxMP).	convert global address to .....	<b>smObjGlobalToLocal()</b>	1243
string to double (ANSI).	convert initial portion of .....	<b>strtod()</b>	1281
number from string to/ address.	convert Internet network .....	<b>inet_network()</b>	679
global address (VxMP).	convert local address to bus.....	<b>sysLocalToBusAdrs()</b>	1306
upper-case equivalent (ANSI).	convert local address to.....	<b>smObjLocalToGlobal()</b>	1245
wide character/ dot notation, store in/ notation, store it in buffer.	convert lower-case letter to.....	<b>toupper()</b>	1392
dotted decimal notation.	convert multibyte character to .....	<b>mbtowc()</b>	805
NTP format.	convert network address from .....	<b>inet_aton()</b>	675
char's to wide char's/ to multibyte char's/ to integer.	convert network address to dot .....	<b>inet_ntoa_b()</b>	680
(ANSI).	convert network address to .....	<b>inet_ntoa()</b>	679
	convert portions of second to .....	<b>sntpsNsecToFraction()</b>	1251
long integer (ANSI).	convert series of multibyte.....	<b>mbstowcs()</b>	804
string (ANSI).	convert series of wide char's .....	<b>wcstombs()</b>	1461
string (POSIX).	convert single-precision value.....	<b>irintf()</b>	718
lower-case equivalent (ANSI).	convert string to double .....	<b>atof()</b>	421
multibyte character/ returns	convert string to int (ANSI) .....	<b>atoi()</b>	422
	convert string to long (ANSI) .....	<b>atol()</b>	422
for presence of floating-point	convert string to long integer .....	<b>strtol()</b>	1284
restore floating-point	convert string to unsigned.....	<b>strtoul()</b>	1285
save floating-point	convert time in seconds into.....	<b>ctime()</b>	510
/floating-point	convert time in seconds into.....	<b>ctime_r()</b>	510
initialize floating-point	convert upper-case letter to .....	<b>tolower()</b>	1392
floating-point	convert wide character to.....	<b>wctomb()</b>	1461
	cookie for device.....	<b>muxTkCookieGet()</b>	897
	coprocessor. probe.....	<b>fppProbe()</b>	601
	coprocessor context.....	<b>fppRestore()</b>	601
	coprocessor context.....	<b>fppSave()</b>	603
	coprocessor support.....	<b>fppArchLib</b>	109
	coprocessor support.....	<b>fppInit()</b>	600
	coprocessor support library .....	<b>fppLib</b>	112
	core memory partition manager.....	<b>memPartLib</b>	160
compute both sine and	cosine.....	<b>sincos()</b>	1228
compute both sine and	cosine.....	<b>sincosf()</b>	1229
compute arc	cosine (ANSI) .....	<b>acos()</b>	405
compute arc	cosine (ANSI) .....	<b>acosf()</b>	406
compute	cosine (ANSI) .....	<b>cos()</b>	500
compute	cosine (ANSI) .....	<b>cosf()</b>	500
compute hyperbolic	cosine (ANSI) .....	<b>cosh()</b>	501
compute hyperbolic	cosine (ANSI) .....	<b>coshf()</b>	501
specify network interface hop	count.....	<b>ifMetricSet()</b>	669
get value of kernel's tick	counter. ....	<b>tickGet()</b>	1379
set value of kernel's tick	counter. ....	<b>tickSet()</b>	1380
half of 64Bit TSC (Timestamp	Counter). get lower .....	<b>pentiumTscGet32()</b>	974
get 64Bit TSC (Timestamp	Counter).....	<b>pentiumTscGet64()</b>	974
reset TSC (Timestamp	Counter).....	<b>pentiumTscReset()</b>	974
install interface packet	counter routine. ....	<b>m2IfPktCountRtnInstall()</b>	779

	Keyword	Name	Page
install interface	counter update routine.....	<b>m2IfCtrUpdateRtnInstall()</b>	775
increment interface	counters. ....	<b>m2IfCounterUpdate()</b>	775
get default values for	counters. ....	<b>m2IfDefaultValsGet()</b>	776
increment interface packet	counters. ....	<b>m2IfGenericPacketCount()</b>	777
get MIB-II RIP-group global	counters. ....	<b>m2RipGlobalCountersGet()</b>	793
PMCs (Performance Monitoring	Counters). show .....	<b>pentiumPmcShow()</b>	970
increment packet	counters for 802.3 device.....	<b>m2If8023PacketCount()</b>	772
create and initialize	counting semaphore. ....	<b>semCCreate()</b>	1176
	counting semaphore library.....	<b>semCLib</b>	264
/and initialize shared memory	counting semaphore (VxMP). ....	<b>semCSmCreate()</b>	1178
cancel currently	counting watchdog. ....	<b>wdCancel()</b>	1464
spy	CPU activity library. ....	<b>spyLib</b>	294
return model name of	CPU board.....	<b>sysModel()</b>	1308
ARM). get	CPU exception vector (PowerPC, .....	<b>excVecGet()</b>	582
ARM). set	CPU exception vector (PowerPC, .....	<b>excVecSet()</b>	584
relinquish	CPU (POSIX).....	<b>sched_yield()</b>	1136
facility/ attach calling	CPU to shared memory objects.....	<b>smObjAttach()</b>	1242
initialize task	CPU utilization tool package. ....	<b>spyLibInit()</b>	1259
type(int/trap), and gate/ get	CPU vector, gate.....	<b>intVecGet2()</b>	699
type(int/trap), and/ set	CPU vector, gate.....	<b>intVecSet2()</b>	703
MIPS, SH, SimSolaris,/ set	CPU vector (trap) (68K, x86,.....	<b>intVecSet()</b>	699
serializing instruction	CPUID. execute .....	<b>pentiumSerialize()</b>	973
get contents of	CR4 register.....	<b>pentiumCr4Get()</b>	952
sets specified value to	CR4 register.....	<b>pentiumCr4Set()</b>	952
to be called at every task	create. add routine .....	<b>taskCreateHookAdd()</b>	1325
queries.	create all types of DNS.....	<b>resolvMkQuery()</b>	1078
symbol table, including group/	create and add symbol to.....	<b>symAdd()</b>	1288
semaphore.	create and initialize binary .....	<b>semBCreate()</b>	1175
semaphore.	create and initialize counting.....	<b>semCCreate()</b>	1176
queue.	create and initialize message.....	<b>msgQCreate()</b>	850
	create and initialize module.....	<b>moduleCreate()</b>	828
mutual-exclusion semaphore.	create and initialize.....	<b>semMCreate()</b>	1184
4.x binary semaphore.	create and initialize release.....	<b>semCreate()</b>	1177
memory binary semaphore/	create and initialize shared.....	<b>semBSmCreate()</b>	1175
memory counting semaphore/	create and initialize shared.....	<b>semCSmCreate()</b>	1178
memory message queue (VxMP).	create and initialize shared.....	<b>msgQSmCreate()</b>	868
device.	create Berkeley Packet Filter.....	<b>bpfDevCreate()</b>	442
service and END.	create binding between network .....	<b>muxBind()</b>	874
calling thread (POSIX).	create cancellation point in.....	<b>pthread_testcancel()</b>	1043
BLK_DEV device.	create CBIO wrapper atop.....	<b>cbioWrapBlkDev()</b>	485
	create <b>cdromFsLib</b> device.....	<b>cdromFsDevCreate()</b>	488
	create disk cache.....	<b>dcacheDevCreate()</b>	514
queue (VxFusion).	create distributed message .....	<b>msgQDistCreate()</b>	851
	create empty ring buffer.....	<b>rngCreate()</b>	1109
	create empty zbuf.....	<b>zbufCreate()</b>	1492
(WindView).	create event-log header .....	<b>wvLogHeaderCreate()</b>	1474
table on disk.	create FDISK-like partition.....	<b>usrFdiskPartCreate()</b>	1421
	create file. ....	<b>creat()</b>	508
event data (Windview).	create file for depositing .....	<b>fileUploadPathCreate()</b>	592
	create file system device.....	<b>dosFsDevCreate()</b>	559

	Keyword	Name	Page
	create hash table.....	hashTblCreate()	646
delete previously added module	create hook routine.....	moduleCreateHookDelete()	829
in system.	create list of all NFS devices.....	nfsDevListGet()	935
	create memory device.....	memDevCreate()	808
multiple files.	create memory device for.....	memDevCreateDir()	810
	create memory partition.....	memPartCreate()	814
	create new line-editor ID.....	ledOpen()	734
context (VxVMI).	create new virtual memory.....	vmContextCreate()	1434
entry.	create or modify ARP table.....	arpAdd()	412
	create pipe device.....	pipeDevCreate()	979
	create private environment.....	envPrivateCreate()	570
	create proxy ARP network.....	proxyNetCreate()	1002
	create pseudo terminal.....	ptyDevCreate()	1043
	create RAM disk device.....	ramDevCreate()	1051
	create remote file device.....	netDevCreate()	911
fixed buffer size.	create remote file device with.....	netDevCreate2()	912
delete previously added task	create routine.....	taskCreateHookDelete()	1326
show list of task	create routines.....	taskCreateHookShow()	1326
initialize device and	create rt11Fs file system.....	rt11FsMkfs()	1127
structure.	create SCSI physical device.....	scsiPhysDevCreate()	1151
	create SCSI sequential device.....	scsiSeqDevCreate()	1158
(VxMP).	create shared memory partition.....	memPartSmCreate()	818
	create symbol table.....	symTblCreate()	1296
(Unimplemented) (ANSI).	create temporary binary file.....	tmpfile()	1391
	create thread (POSIX).....	pthread_create()	1025
key (POSIX).	create thread specific data.....	pthread_key_create()	1029
suitable for use with dosFs.	create TrueFFS block device.....	tffsDevCreate()	1363
	create UNIX disk device.....	unixDiskDevCreate()	1414
serial channel.	create VxWorks device for.....	ttyDevCreate()	1405
	create watchdog timer.....	wdCreate()	1465
send it to TCP socket.	create zbuf from user data and.....	zbufSockBufSend()	1502
and send it to UDP socket.	create zbuf from user message.....	zbufSockBufSendto()	1503
buffer and insert into zbuf.	create zbuf segment from.....	zbufInsertBuf()	1497
(PowerPC/ connect C routine to	critical exception vector.....	excCrtConnect()	578
(PowerPC/ connect C routine to	critical interrupt vector.....	excIntCrtConnect()	581
ANSI	ctype documentation.....	ansiCtype	14
compute	cube root.....	cbrt()	485
compute	cube root.....	cbrtf()	486
clear line from	CY7C604 cache.....	cacheCy604ClearLine()	450
clear page from	CY7C604 cache.....	cacheCy604ClearPage()	450
clear region from	CY7C604 cache.....	cacheCy604ClearRegion()	451
clear segment from	CY7C604 cache.....	cacheCy604ClearSegment()	451
initialize Cypress	CY7C604 cache library.....	cacheCy604LibInit()	452
initialize	Cypress CY7C604 cache library.....	cacheCy604LibInit()	452
return contents of register	d0 (also d1 - d7) (68K).....	d0()	512
contents of register d0 (also	d1 - d7) (68K). return.....	d0()	512
of register d0 (also d1 -	d7) (68K). return contents.....	d0()	512
VxWorks remote login	daemon.....	rlogind()	1105
initialize mount	daemon.....	mountdInit()	836
TFTP server	daemon task.....	tftpdTask()	1371

	Keyword	Name	Page
display distributed name	database filtered by type/ .....	<b>distNameFilterShow()</b>	552
distributed name	database library (VxFusion). .....	<b>distNameLib</b>	84
shared memory objects name	database library (VxMP). .....	<b>smNameLib</b>	284
(VxFusion). distributed name	database show routines.....	<b>distNameShow</b>	85
shared memory objects name	database show routines (VxMP). .....	<b>smNameShow</b>	286
add entry to distributed name	database (VxFusion). .....	<b>distNameAdd()</b>	551
find object by name in local	database (VxFusion). .....	<b>distNameFind()</b>	553
entry from distributed name	database (VxFusion). remove.....	<b>distNameRemove()</b>	555
entire distributed name	database (VxFusion). display .....	<b>distNameShow()</b>	555
add name to shared memory name	database (VxMP). .....	<b>smNameAdd()</b>	1237
shared memory objects name	database (VxMP). /object from.....	<b>smNameRemove()</b>	1240
of shared memory objects name	database (VxMP). /contents.....	<b>smNameShow()</b>	1240
set rt11Fs file system	date.....	<b>rt11FsDateSet()</b>	1125
enable last access	date updating for this volume. ....	<b>dosFsLastAccessDateEnable()</b>	560
events.	deactivate specific network .....	<b>wvNetEventDisable()</b>	1477
run-time support for memory	deallocation (C++). default.....	<b>operator delete()</b>	946
routines. set	debug level of ftp library .....	<b>ftpLibDebugOptionSet()</b>	626
routines. set	debug level of <b>netDrv</b> library .....	<b>netDrvDebugLevelSet()</b>	913
partition. set	debug options for memory .....	<b>memPartOptionsSet()</b>	816
memory system partition/ set	debug options for shared .....	<b>smMemOptionsSet()</b>	1234
memory partition. set	debug options for system .....	<b>memOptionsSet()</b>	812
get content of	Debug Register 0 to 7 (x86).....	<b>vxDrGet()</b>	1450
set value to	Debug Register 0 to 7 (x86). .....	<b>vxDrSet()</b>	1451
architecture-dependent	debugger library. ....	<b>dbgArchLib</b>	63
	debugging facilities. ....	<b>dbgLib</b>	65
display	debugging help menu. ....	<b>dbgHelp()</b>	513
protocol. display	debugging information for TCP .....	<b>tcpDebugShow()</b>	1357
specify amount of	debugging output. ....	<b>ripDebugLevelSet()</b>	1092
initialize local	debugging package.....	<b>dbgInit()</b>	514
test whether character is	decimal digit (ANSI). ....	<b>isdigit()</b>	721
network address to dotted	decimal notation. convert .....	<b>inet_ntoa()</b>	679
function. replace	default address resolution .....	<b>muxAddrResFuncAdd()</b>	871
print current	default directory. ....	<b>pwd()</b>	1048
change	default directory. ....	<b>cd()</b>	486
fds. set shell's	default input/output/error .....	<b>shellOrigStdSet()</b>	1214
spawn task with	default parameters. ....	<b>sp()</b>	1255
routine.	default password encryption .....	<b>loginDefaultEncrypt()</b>	751
set current	default path.....	<b>chdir()</b>	491
get current	default path.....	<b>getwd()</b>	640
get current	default path.....	<b>ioDefPathGet()</b>	705
set current	default path.....	<b>ioDefPathSet()</b>	706
get current	default path (POSIX). ....	<b>getcwd()</b>	636
memory deallocation (C++).	default run-time support for .....	<b>operator delete()</b>	946
operator new (C++).	default run-time support for .....	<b>operator new()</b>	947
operator new (nothrow) (C++).	default run-time support for .....	<b>operator new()</b>	947
set	default task ID. ....	<b>taskIdDefault()</b>	1331
(MIPS). initialize	default task status register.....	<b>taskSRInit()</b>	1346
get	default values for counters. ....	<b>m2IfDefaultValsGet()</b>	776
time.	delay for specified amount of .....	<b>sleep()</b>	1231
nanoseconds.	delay for specified number of .....	<b>sysNanoDelay()</b>	1308

	Keyword	Name	Page
	get	delay on polling task.....	<b>muxTaskDelayGet()</b> 893
	set inter-cycle	delay on polling task.....	<b>muxTaskDelaySet()</b> 894
		delay task from executing .....	<b>taskDelay()</b> 1327
	to be called at every task	delete. add routine .....	<b>taskDeleteHookAdd()</b> 1329
	function.	delete address resolution .....	<b>muxAddrResFuncDel()</b> 872
		delete all breakpoints.....	<b>bdall()</b> 428
	groups.	delete all MIB-II library .....	<b>m2Delete()</b> 770
	access ICMP group.	delete all resources used to .....	<b>m2IcmpDelete()</b> 771
	access interface group.	delete all resources used to .....	<b>m2IfDelete()</b> 776
	access IP group.	delete all resources used to .....	<b>m2IpDelete()</b> 789
	access TCP group.	delete all resources used to .....	<b>m2TcpDelete()</b> 798
	access UDP group.	delete all resources used to .....	<b>m2UdpDelete()</b> 800
	with network interface.	delete all routes associated.....	<b>ifAllRoutesDelete()</b> 660
	from list.	delete and return first node .....	<b>lstGet()</b> 765
	segment from module. get	(delete and return) first.....	<b>moduleSegGet()</b> 835
		delete breakpoint.....	<b>bd()</b> 427
		delete bytes from zbuf. ....	<b>zbufCut()</b> 1492
		delete device from I/O system.....	<b>iosDevDelete()</b> 708
	list.	delete directory from access.....	<b>tftpdDirectoryRemove()</b> 1369
	queue from group (VxFusion).	delete distributed message .....	<b>msgQDistGrpDelete()</b> 854
	address record.	delete Ethernet multicast.....	<b>etherMultiDel()</b> 574
	authentication key.	delete existing RIP.....	<b>ripAuthKeyDelete()</b> 1088
		delete file (POSIX). ....	<b>unlink()</b> 1418
		delete hash table. ....	<b>hashTblDelete()</b> 647
	wildcards.	delete hierarchy of files with .....	<b>xdelete()</b> 1490
	basis.	delete hook routine on unit.....	<b>pppHookDelete()</b> 983
		delete host from host table.....	<b>hostDelete()</b> 652
	network interface.	delete interface address for .....	<b>ifAddrDelete()</b> 659
	exclusion list.	delete interface from RIP.....	<b>ripIfExcludeListDelete()</b> 1094
		delete IP filter hook routine. ....	<b>ipFilterHookDelete()</b> 716
	routine.	delete lease data storage.....	<b>dhcpcCacheHookDelete()</b> 524
		delete logging fd.....	<b>logFdDelete()</b> 750
		delete memory device.....	<b>memDevDelete()</b> 810
		delete memory pool. ....	<b>netPoolDelete()</b> 923
		delete message queue. ....	<b>msgQDelete()</b> 851
	add, modify, or	delete MIB-II ARP entry. ....	<b>m2IpAtransTblEntrySet()</b> 788
	(use <b>unld()</b> to reclaim/	delete module ID information.....	<b>moduleDelete()</b> 830
	device's multicast table.	delete multicast address from .....	<b>muxMCastAddrDel()</b> 883
	wake-up list. find and	delete node from <b>select()</b> .....	<b>selNodeDelete()</b> 1171
		delete pipe device.....	<b>pipeDevDelete()</b> 979
		delete PPP network interface.....	<b>pppDelete()</b> 982
	create hook routine.	delete previously added module ....	<b>moduleCreateHookDelete()</b> 829
	create routine.	delete previously added task.....	<b>taskCreateHookDelete()</b> 1326
	delete routine.	delete previously added task.....	<b>taskDeleteHookDelete()</b> 1329
	switch routine.	delete previously added task.....	<b>taskSwitchHookDelete()</b> 1350
		delete proxy network .....	<b>proxyNetDelete()</b> 1002
	access MIB-II system group.	delete resources used to .....	<b>m2SysDelete()</b> 795
		delete ring buffer. ....	<b>rngDelete()</b> 1109
		delete RIP MIB support.....	<b>m2RipDelete()</b> 792
		delete route.....	<b>routeDelete()</b> 1116

	Keyword	Name	Page
	table. delete route from routing.....	<b>mRouteDelete()</b>	847
	table. delete route from routing.....	<b>mRouteEntryDelete()</b>	849
	network interface. delete routes associated with .....	<b>ifRouteDelete()</b>	670
	delete previously added task delete routine.....	<b>taskDeleteHookDelete()</b>	1329
	show list of task delete routines.....	<b>taskDeleteHookShow()</b>	1330
	structure. delete SCSI physical-device.....	<b>scsiPhysDevDelete()</b>	1152
	authentication secrets table. delete secret from PPP.....	<b>pppSecretDelete()</b>	994
	delete semaphore.....	<b>semDelete()</b>	1179
	(POSIX). delete signal from signal set.....	<b>sigdelset()</b>	1218
	list. delete specified node from .....	<b>lstDelete()</b>	763
	delete symbol table.....	<b>symTblDelete()</b>	1296
	delete task.....	<b>taskDelete()</b>	1327
	delete task.....	<b>td()</b>	1358
	restriction. delete task without .....	<b>taskDeleteForce()</b>	1328
	key (POSIX). delete thread specific data .....	<b>pthread_key_delete()</b>	1029
	list. delete trigger from trigger.....	<b>trgDelete()</b>	1395
	table. delete user entry from login.....	<b>loginUserDelete()</b>	755
	(VxVMI). delete virtual memory context.....	<b>vmContextDelete()</b>	1435
	delete watchdog timer.....	<b>wdDelete()</b>	1465
	delete zbuf.....	<b>zbufDelete()</b>	1493
	make calling task safe from deletion.....	<b>taskSafe()</b>	1342
	make calling task unsafe from deletion.....	<b>taskUnsafe()</b>	1352
	change C++ demangling mode (C++).....	<b>cplusDemanglerSet()</b>	504
	change C++ demangling style (C++).....	<b>cplusDemanglerStyleSet()</b>	505
	<b>muxTkPollReceive()</b> . now deprecated, see .....	<b>muxPollReceive()</b>	889
	<b>muxTkPollSend()</b> . now deprecated, see .....	<b>muxPollSend()</b>	890
	get current interrupt nesting depth.....	<b>intCount()</b>	686
	call static destructors (C++).....	<b>cplusDtors()</b>	505
	call all linked static destructors (C++).....	<b>cplusDtorsLink()</b>	506
	thread/ get value of detachstate attribute in.....	<b>pthread_attr_getdetachstate()</b>	1008
	thread attributes object/ set detachstate attribute in.....	<b>pthread_attr_setdetachstate()</b>	1013
	detect change in media.....	<b>scsiSeqStatusCheck()</b>	1160
	create RAM disk device.....	<b>ramDevCreate()</b>	1051
	initialize RAM Disk device.....	<b>ramDiskDevCreate()</b>	1052
	read bytes from file or device.....	<b>read()</b>	1064
	on specified physical device. /BLK_DEV structures.....	<b>scsiBlkDevShow()</b>	1139
	issue ERASE command to SCSI device.....	<b>scsiErase()</b>	1142
	command to SCSI device. issue FORMAT_UNIT.....	<b>scsiFormatUnit()</b>	1142
	issue INQUIRY command to SCSI device.....	<b>scsiInquiry()</b>	1144
	command to SCSI device. issue LOAD/UNLOAD .....	<b>scsiLoadUnit()</b>	1145
	command to SCSI device. issue MODE_SELECT.....	<b>scsiModeSelect()</b>	1149
	MODE_SENSE command to SCSI device. issue.....	<b>scsiModeSense()</b>	1149
	information for physical device. show status.....	<b>scsiPhysDevShow()</b>	1153
	bytes or blocks from SCSI tape device. read .....	<b>scsiRdTape()</b>	1154
	command to SCSI device. issue READ_CAPACITY .....	<b>scsiReadCapacity()</b>	1155
	issue RELEASE command to SCSI device.....	<b>scsiRelease()</b>	1155
	command to SCSI device. issue RELEASE UNIT .....	<b>scsiReleaseUnit()</b>	1156
	issue RESERVE command to SCSI device.....	<b>scsiReserve()</b>	1157
	command to SCSI device. issue RESERVE UNIT .....	<b>scsiReserveUnit()</b>	1157
	issue REWIND command to SCSI device.....	<b>scsiRewind()</b>	1158

	Keyword	Name	Page
create SCSI sequential	device. ....	scsiSeqDevCreate()	1158
command to SCSI	device. /READ_BLOCK_LIMITS .....	scsiSeqReadBlockLimits()	1159
on specified physical SCSI	device. move tape.....	scsiSpace()	1161
command to SCSI	device. issue START_STOP_UNIT.....	scsiStartStopUnit()	1162
command to SCSI tape	device. issue MODE_SELECT .....	scsiTapeModeSelect()	1163
command to SCSI tape	device. issue MODE_SENSE .....	scsiTapeModeSense()	1163
command to SCSI	device. issue TEST_UNIT_READY.....	scsiTestUnitRdy()	1166
file marks to SCSI sequential	device. write.....	scsiWrtFileMarks()	1167
write data to SCSI tape	device. ....	scsiWrtTape()	1168
to boot-image region of flash	device. write .....	tffsBootImagePut()	1362
do task-level read for tty	device. ....	tyRead()	1411
do task-level write for tty	device. ....	tyWrite()	1412
create UNIX disk	device. ....	unixDiskDevCreate()	1414
create Berkeley Packet Filter	device. ....	bpfDevCreate()	442
destroy Berkeley Packet Filter	device. ....	bpfDevDelete()	442
perform ioctl operation on	device. ....	cbioIoctl()	479
return mode setting for CBIO	device. ....	cbioModeGet()	481
set mode for CBIO	device. ....	cbioModeSet()	481
determine ready status of CBIO	device. ....	cbioRdyChgdGet()	482
change in ready status of CBIO	device. force.....	cbioRdyChgdSet()	483
print information about CBIO	device. ....	cbioShow()	484
CBIO wrapper atop BLK_DEV	device. create.....	cbioWrapBlkDev()	485
create cdromFsLib	device. ....	cdromFsDevCreate()	488
disable disk cache for this	device. ....	dcacheDevDisable()	515
set new size to disk cache	device. ....	dcacheDevMemResize()	516
create file system	device. ....	dosFsDevCreate()	559
underlying driver is tty	device. return whether .....	isatty()	720
packet counters for 802.3	device. increment .....	m2If8023PacketCount()	772
create memory	device. ....	memDevCreate()	808
delete memory	device. ....	memDevDelete()	810
information to MUX or to	device. send control .....	muxIoctl()	880
returns cookie for	device. ....	muxTkCookieGet()	897
network service from specified	device. detach .....	muxUnbind()	904
create remote file	device. ....	netDevCreate()	911
information from requested NFS	device. read configuration .....	nfsDevInfoGet()	934
unmount NFS	device. ....	nfsUnmount()	943
create pipe	device. ....	pipeDevCreate()	979
delete pipe	device. ....	pipeDevDelete()	979
channels. provide terminal	device access to serial .....	ttyDrv	326
system. initialize	device and create rt11Fs file.....	rt11FsMkfs()	1127
command to SCSI	device and read results. /REQUEST_SENSE.....	scsiReqSense()	1156
channel. get SIO_CHAN	device associated with serial .....	sysSerialChanGet()	1315
routine. start	device by calling start .....	muxDevStart()	877
routine. stop	device by calling stop .....	muxDevStop()	878
handle	device control requests. ....	tyIoctl()	1409
initialize rt11Fs	device descriptor. ....	rt11FsDevInit()	1126
initialize tty	device descriptor. ....	tyDevInit()	1408
remove tty	device descriptor. ....	tyDevRemove()	1408
pseudo-memory	device driver.....	memDrv	156
extract backplane address from	device field. ....	bootBpAnchorExtract()	432

	Keyword	Name	Page
	tape sequential device file system library. ....	<b>tapeFsLib</b>	302
	create memory device for multiple files. ....	<b>memDevCreateDir()</b>	810
	create VxWorks device for serial channel. ....	<b>ttyDevCreate()</b>	1405
	format flash device for use with TrueFFS. ....	<b>tffsDevFormat()</b>	1364
	delete device from I/O system. ....	<b>iosDevDelete()</b>	708
tMuxPollTask. removes	device from list polled by .....	<b>muxPollDevDel()</b>	887
unloads	device from MUX. ....	<b>muxDevUnload()</b>	879
initialize CBIO	device (Generic). ....	<b>cbioDevCreate()</b>	478
find I/O	device in device list. ....	<b>iosDevFind()</b>	709
socket interfaces. show	device information on all .....	<b>tffsShowAll()</b>	1368
socket interface. show	device information on specific .....	<b>tffsShow()</b>	1367
connect BSP serial	device interrupts. ....	<b>sysSerialHwInit2()</b>	1316
MUX. tests whether	device is already loaded into .....	<b>muxDevExists()</b>	876
interface. checks if	device is NPT or END .....	<b>muxTkDrvCheck()</b>	898
tMuxPollTask. reports whether	device is on list polled by .....	<b>muxPollDevStat()</b>	888
SCSI sequential access	device library (SCSI-2) .....	<b>scsiSeqLib</b>	260
find I/O device in	device list. ....	<b>iosDevFind()</b>	709
descriptor pointer. convert	device name into DOS volume .....	<b>dosFsVolDescGet()</b>	561
/structure with CBIO	device parameters. ....	<b>cbioParamsGet()</b>	482
obtain CBIO	device semaphore. ....	<b>cbioLock()</b>	480
release CBIO	device semaphore. ....	<b>cbioUnlock()</b>	484
create SCSI physical	device structure. ....	<b>scsiPhysDevCreate()</b>	1151
add	device to I/O system. ....	<b>iosDevAdd()</b>	708
tMuxPollTask. adds	device to list polled by .....	<b>muxPollDevAdd()</b>	887
find	device using string name. ....	<b>endFindByName()</b>	569
modify mode of raw	device volume. ....	<b>rawFsModeChange()</b>	1055
disable raw	device volume. ....	<b>rawFsVolUnmount()</b>	1056
disable tape	device volume. ....	<b>tapeFsVolUnmount()</b>	1322
create remote file	device with fixed buffer size. ....	<b>netDevCreate2()</b>	912
system functions. associate	device with ntPassFs file. ....	<b>ntPassFsDevInit()</b>	943
functions. associate	device with passFs file system .....	<b>passFsDevInit()</b>	949
associate sequential	device with tape volume/ .....	<b>tapeFsDevInit()</b>	1320
function for sequential access	devices. perform I/O control .....	<b>scsiSeqIoctl()</b>	1159
list all system-known	devices. ....	<b>devs()</b>	519
display mounted NFS	devices. ....	<b>nfsDevShow()</b>	935
cache-safe buffer for DMA	devices and drivers. allocate .....	<b>cacheDmaMalloc()</b>	453
to boot ROMs. reset network	devices and transfer control .....	<b>reboot()</b>	1066
controller. list physical	devices attached to SCSI .....	<b>scsiShow()</b>	1160
controller. configure all	devices connected to SCSI .....	<b>scsiAutoConfig()</b>	1137
display list of	devices in system. ....	<b>iosDevShow()</b>	709
create list of all NFS	devices in system. ....	<b>nfsDevListGet()</b>	935
add multicast address to	device's multicast table. ....	<b>muxMCastAddrAdd()</b>	882
delete multicast address from	device's multicast table. ....	<b>muxMCastAddrDel()</b>	883
display configuration of	devices registered with MUX. ....	<b>muxShow()</b>	893
common commands for all	devices (SCSI-2). /library .....	<b>scsiCommonLib</b>	256
SCSI library for direct access	devices (SCSI-2) .....	<b>scsiDirectLib</b>	257
/down tMuxPollTask and returns	devices to interrupt mode. ....	<b>muxPollEnd()</b>	888
initialize BSP serial	devices to quiescent state. ....	<b>sysSerialHwInit()</b>	1315
reset all SIO	devices to quiet state. ....	<b>sysSerialReset()</b>	1316
function. perform	device-specific I/O control .....	<b>scsiIoctl()</b>	1145

	Keyword	Name	Page
configuration parameters with	DHCP. obtain set of network .....	<b>dhcpcBind()</b>	520
configuration parameters with	DHCP. obtain additional .....	<b>dhcpcBootInformGet()</b>	521
configuration parameters with	DHCP. obtain additional .....	<b>dhcpcInformGet()</b>	526
initialize network with	DHCP at boot time .....	<b>dhcpcBootBind()</b>	521
code library.	DHCP boot-time client library .....	<b>dhcpcBootLib</b>	71
disable	DHCP client interface shared .....	<b>dhcpcCommonLib</b>	72
initialization.	DHCP client library .....	<b>dhcpcShutdown()</b>	536
data structures. set up	DHCP client library .....	<b>dhcpcLibInit()</b>	528
	DHCP client parameters and .....	<b>dhcpcBootInit()</b>	522
	DHCP relay agent library .....	<b>dhcprLib</b>	75
/Host Configuration Protocol	(DHCP) run-time client API .....	<b>dhcpcLib</b>	73
information display routines.	DHCP run-time client .....	<b>dhcpcShow</b>	75
retrieve current	DHCP server. ....	<b>dhcpcServerGet()</b>	535
display current	DHCP server. ....	<b>dhcpcServerShow()</b>	535
/Host Configuration Protocol	(DHCP) server library .....	<b>dhcpsLib</b>	76
data structures. set up	DHCP server parameters and .....	<b>dhcpsInit()</b>	540
initialize	DHCP show facility .....	<b>dhcpsShowInit()</b>	536
(ANSI). put	diagnostics into programs .....	<b>assert()</b>	418
calendar times/ compute	difference between two .....	<b>difftime()</b>	542
whether character is decimal	digit (ANSI). test .....	<b>isdigit()</b>	721
character is hexadecimal	digit (ANSI). test whether .....	<b>isxdigit()</b>	725
RST line on SCSI bus (Western	Digital WD33C93 only). assert .....	<b>sysScsiBusReset()</b>	1312
(SCSI-2). SCSI library for	direct access devices .....	<b>scsiDirectLib</b>	257
print current default	directory. ....	<b>pwd()</b>	1048
remove	directory. ....	<b>rmdir()</b>	1107
file attributes on file or	directory. modify MS-DOS .....	<b>attrib()</b>	423
change default	directory. ....	<b>cd()</b>	486
generate brief listing of	directory. ....	<b>ls()</b>	759
make	directory. ....	<b>mkdir()</b>	821
mv file into other	directory. ....	<b>mv()</b>	905
do long listing of	directory and all/ .....	<b>llr()</b>	738
list contents of	directory and any of/ .....	<b>lsr()</b>	761
generate long listing of	directory contents .....	<b>ll()</b>	737
list	directory contents via FTP .....	<b>ftplS()</b>	627
(POSIX). open	directory for searching .....	<b>opendir()</b>	946
delete	directory from access list .....	<b>tftpdDirectoryRemove()</b>	1369
(POSIX). directory	handling library .....	<b>dirLib</b>	81
list contents of	directory (multi-purpose) .....	<b>dirList()</b>	542
read one entry from	directory (POSIX) .....	<b>readdir()</b>	1064
reset position to start of	directory (POSIX) .....	<b>rewinddir()</b>	1082
close	directory (POSIX) .....	<b>closedir()</b>	497
add	directory to access list .....	<b>tftpdDirectoryAdd()</b>	1369
hardware snooping of caches is	disabled. inform SCSI that .....	<b>scsiCacheSnoopDisable()</b>	1140
specified number of/	disassemble and display .....	<b>l()</b>	729
removed interfaces for router	discovery. check for new or .....	<b>rdiscIfReset()</b>	1062
initialize router	discovery .....	<b>rdiscLibInit()</b>	1063
implement ICMP router	discovery control function .....	<b>rdcIf()</b>	1059
implement ICMP router	discovery function .....	<b>rdisc()</b>	1061
initialize ICMP router	discovery function .....	<b>rdiscInit()</b>	1062
ICMP router	discovery server library .....	<b>rdiscLib</b>	230

	Keyword	Name	Page
DOS file system from floppy	disk. mount .....	usrFdConfig()	1420
FDISK-like partition table on	disk. create .....	usrFdiskPartCreate()	1421
DOS file system from IDE hard	disk. mount .....	usrIdeConfig()	1424
format	disk .....	diskFormat()	543
initialize partitioned	disk .....	dpartDevCreate()	564
create	disk cache .....	dcacheDevCreate()	514
re-enable	disk cache .....	dcacheDevEnable()	516
print information about	disk cache .....	dcacheShow()	519
set new size to	disk cache device .....	dcacheDevMemResize()	516
	disk cache driver .....	dcacheCbio	67
disable	disk cache for this device .....	dcacheDevDisable()	515
modify tunable	disk cache parameters .....	dcacheDevTune()	517
RAM	Disk Cached Block Driver .....	ramDiskCbio	225
create RAM	disk device .....	ramDevCreate()	1051
initialize RAM	Disk device .....	ramDiskDevCreate()	1052
create UNIX	disk device .....	unixDiskDevCreate()	1414
install UNIX	disk driver .....	unixDrv()	1416
initialize hard	disk driver .....	usrAtaInit()	1420
RAM	disk driver .....	ramDrv	225
(optional). prepare RAM	disk driver for use .....	ramDrv()	1053
and VxSim for HP). UNIX-file	disk driver (VxSim for Solaris .....	unixDrv	332
floppy	disk initialization .....	usrFd	336
initialize dosFs	disk on top of UNIX .....	unixDiskInit()	1415
DOS file system from ATA hard	disk or CDROM. mount .....	usrAtaConfig()	1419
generic	disk partition manager .....	dpartCbio	98
disable superscalar	dispatch (MC68060) .....	vxSSDisable()	1458
enable superscalar	dispatch (MC68060) .....	vxSSEnable()	1458
group (VxFusion). delete	distributed message queue from .....	msgQDistGrpDelete()	854
group library (VxFusion).	distributed message queue .....	msgQDistGrpLib	180
group show routines/	distributed message queue .....	msgQDistGrpShow	181
package/ initialize	distributed message queue show .....	msgQDistShowInit()	859
routines (VxFusion).	distributed message queue show .....	msgQDistShow	182
group (VxFusion). add	distributed message queue to .....	msgQDistGrpAdd()	853
(VxFusion). create	distributed message queue .....	msgQDistCreate()	851
get number of messages in	distributed message queue/ .....	msgQDistNumMsgs()	855
receive message from	distributed message queue/ .....	msgQDistReceive()	856
(VxFusion). send message to	distributed message queue .....	msgQDistSend()	857
filtered by type/ display	distributed name database .....	distNameFilterShow()	552
library (VxFusion).	distributed name database .....	distNameLib	84
routines (VxFusion).	distributed name database show .....	distNameShow	85
(VxFusion). add entry to	distributed name database .....	distNameAdd()	551
(VxFusion). remove entry from	distributed name database .....	distNameRemove()	555
(VxFusion). display entire	distributed name database .....	distNameShow()	555
function (VxFusion). perform	distributed objects control .....	distCtl()	544
initialization and control/	distributed objects .....	distLib	83
adapter show routines/	distributed objects interface .....	distIfShow	83
queue library (VxFusion).	distributed objects message .....	msgQDistLib	181
buffer library (VxFusion).	distributed objects telegram .....	distTBufLib	85
quotient and remainder of	division (ANSI). compute .....	ldiv()	732
allocate cache-safe buffer for	DMA devices and drivers .....	cacheDmaMalloc()	453

	Keyword	Name	Page
packet. expand	DNS compressed name from DNS .....	<b>resolvDNExpand()</b>	1074
compress	DNS name in DNS packet.....	<b>resolvDNComp()</b>	1074
compress DNS name in	DNS packet.....	<b>resolvDNComp()</b>	1074
DNS compressed name from	DNS packet. expand .....	<b>resolvDNExpand()</b>	1074
create all types of	DNS queries.....	<b>resolvMkQuery()</b>	1078
	DNS resolver library.....	<b>resolvLib</b>	232
address. query	DNS server for host name of IP.....	<b>resolvGetHostByAddr()</b>	1075
host. query	DNS server for IP address of .....	<b>resolvGetHostByName()</b>	1076
inflate code using public	domain zlib functions.....	<b>inflateLib</b>	123
disk or CDRom. mount	DOS file system from ATA hard.....	<b>usrAtaConfig()</b>	1419
disk. mount	DOS file system from floppy .....	<b>usrFdConfig()</b>	1420
disk. mount	DOS file system from IDE hard.....	<b>usrIdeConfig()</b>	1424
convert device name into	DOS volume descriptor pointer.....	<b>dosFsVolDescGet()</b>	561
suitable for use with	dosFs. /TrueFFS block device .....	<b>tfFsDevCreate()</b>	1363
initialize	dosFs disk on top of UNIX.....	<b>unixDiskInit()</b>	1415
prepare to use	dosFs library.....	<b>dosFsLibInit()</b>	560
data. display	dosFs volume configuration .....	<b>dosFsShow()</b>	561
return very large	double.....	<b>infinity()</b>	681
initial portion of string to	double (ANSI). convert .....	<b>strtod()</b>	1281
convert string to	double (ANSI).....	<b>atof()</b>	421
integer. convert	double-precision value to.....	<b>rint()</b>	717
library.	doubly linked list subroutine .....	<b>lstLib</b>	137
initialize pseudo-terminal	driver.....	<b>ptyDrv()</b>	1044
AIO system	driver.....	<b>aioSysDrv</b>	13
initialize tty	driver.....	<b>ttyDrv()</b>	1405
install UNIX disk	driver.....	<b>unixDrv()</b>	1416
initialize hard disk	driver.....	<b>usrAtaInit()</b>	1420
pseudo-memory device	driver.....	<b>memDrv</b>	156
network remote file I/O	driver.....	<b>netDrv</b>	190
Network File System (NFS) I/O	driver.....	<b>nfsDrv</b>	195
pipe I/O	driver.....	<b>pipeDrv</b>	210
pseudo-terminal	driver.....	<b>ptyDrv</b>	223
RAM Disk Cached Block	Driver.....	<b>ramDiskCbio</b>	225
RAM disk	driver.....	<b>ramDrv</b>	225
memory network (backplane)	driver. /interface to shared.....	<b>smNetLib</b>	287
initialize AIO system	driver.....	<b>aioSysInit()</b>	407
initialize BPF	driver.....	<b>bpfDrv()</b>	443
of multicast addresses from	driver. retrieve table.....	<b>etherMultiGet()</b>	574
disk cache	driver.....	<b>dcacheCbio</b>	67
install I/O	driver.....	<b>iosDrvInstall()</b>	710
remove I/O	driver.....	<b>iosDrvRemove()</b>	710
install memory	driver.....	<b>memDrv()</b>	811
bind NPT protocol to	driver.....	<b>muxTkBind()</b>	895
for packet from NPT or END	driver. poll .....	<b>muxTkPollReceive()</b>	898
receive packet from NPT	driver.....	<b>muxTkReceive()</b>	901
install network remote file	driver.....	<b>netDrv()</b>	913
install NFS	driver.....	<b>nfsDrv()</b>	937
system driver number for NFS	driver. return IO.....	<b>nfsDrvNumGet()</b>	937
initialize pipe	driver.....	<b>pipeDrv()</b>	980
prepare RAM disk	driver for use (optional).....	<b>ramDrv()</b>	1053

	Keyword	Name	Page
	load driver into MUX.....	<b>muxDevLoad()</b>	876
return whether underlying	driver is tty device.....	<b>isatty()</b>	720
Packet Filter (BPF) I/O	driver library. Berkeley.....	<b>bpfDrv</b>	35
return IO system	driver number for NFS driver.....	<b>nfsDrvNumGet()</b>	937
shared memory network	driver show routines.....	<b>smNetShow</b>	288
tty	driver support library.....	<b>tyLib</b>	326
VxSim for HP). UNIX-file disk	driver (VxSim for Solaris and.....	<b>unixDrv</b>	332
buffer for DMA devices and	drivers. allocate cache-safe.....	<b>cacheDmaMalloc()</b>	453
flush data cache for	drivers.....	<b>cacheDrvFlush()</b>	454
invalidate data cache for	drivers.....	<b>cacheDrvInvalidate()</b>	454
translate physical address for	drivers.....	<b>cacheDrvPhysToVirt()</b>	455
translate virtual address for	drivers.....	<b>cacheDrvVirtToPhys()</b>	455
translate physical address for	drivers.....	<b>cacheTiTms390PhysToVirt()</b>	474
display list of system	drivers.....	<b>iosDrvShow()</b>	711
validate open fd and return	driver-specific value.....	<b>iosFdValue()</b>	711
print contents of task's	DSP registers.....	<b>dspTaskRegsShow()</b>	566
initialize	DSP show facility.....	<b>dspShowInit()</b>	566
	dsp show routines.....	<b>dspShow</b>	100
initialize	DSP support.....	<b>dspInit()</b>	565
	dsp support library.....	<b>dspLib</b>	100
	duplicate mBlk.....	<b>netMblkDup()</b>	920
	duplicate mBlk chain.....	<b>netMblkChainDup()</b>	916
	duplicate zbuf.....	<b>zbufDup()</b>	1494
Protocol (DHCP) run-time/	Dynamic Host Configuration.....	<b>dhcpcLib</b>	73
Protocol (DHCP) server/	Dynamic Host Configuration.....	<b>dhcpsLib</b>	76
(POSIX).	dynamic package initialization.....	<b>pthread_once()</b>	1038
library.	dynamic ring buffer (rBuff).....	<b>rBuffLib</b>	230
of register edi (also esi -	eax) (x86/SimNT). /contents.....	<b>edi()</b>	568
return contents of register	edi (also esi - eax)/.....	<b>edi()</b>	568
get content of	EFLAGS register (x86).....	<b>vxEflagsGet()</b>	1451
set value to	EFLAGS register (x86).....	<b>vxEflagsSet()</b>	1452
high-level floating-point	emulation library.....	<b>mathSoftLib</b>	155
default password	encryption routine.....	<b>loginDefaultEncrypt()</b>	751
install	encryption routine.....	<b>loginEncryptInstall()</b>	751
for stream (ANSI). clear	end-of-file and error flags.....	<b>clearerr()</b>	493
change	end-of-file character.....	<b>tyEOFSet()</b>	1409
stream (ANSI). test	end-of-file indicator for.....	<b>feof()</b>	588
create private	environment.....	<b>envPrivateCreate()</b>	570
destroy private	environment.....	<b>envPrivateDestroy()</b>	570
goto by restoring saved	environment (ANSI). /non-local.....	<b>longjmp()</b>	759
display	environment for task.....	<b>envShow()</b>	571
argument (ANSI). save calling	environment in jmp_buf.....	<b>setjmp()</b>	1201
set	environment variable.....	<b>putenv()</b>	1046
get	environment variable (ANSI).....	<b>getenv()</b>	636
initialize	environment variable facility.....	<b>envLibInit()</b>	569
	environment variable library.....	<b>envLib</b>	101
issue	ERASE command to SCSI device.....	<b>scsiErase()</b>	1142
map error number in	errno to error message (ANSI).....	<b>perror()</b>	976
standard output or standard	error. set line buffering for.....	<b>setlinebuf()</b>	1202
probe address for bus	error.....	<b>vxMemProbe()</b>	1454

	Keyword	Name	Page
	clear end-of-file and	error flags for stream (ANSI).....	<b>clearerr()</b> 493
	/(take) semaphore, returning	error if unavailable (POSIX).....	<b>sem_trywait()</b> 1194
	pointer (ANSI). test	error indicator for file.....	<b>ferror()</b> 589
	store buffer after data store	error interrupt. clean up.....	<b>cleanUpStoreBuffer()</b> 493
	log formatted	error message.....	<b>logMsg()</b> 757
	map error number in errno to	error message (ANSI).....	<b>perror()</b> 976
	message (ANSI). map	error number in errno to error.....	<b>perror()</b> 976
	(ANSI). map	error number to error string.....	<b>strerror()</b> 1274
	(POSIX). map	error number to error string.....	<b>strerror_r()</b> 1274
		error status library.....	<b>errnoLib</b> 102
	I/O operation/ retrieve	error status of asynchronous.....	<b> aio_error()</b> 408
	print definition of specified	error status value.....	<b>printErrno()</b> 996
	task. get	error status value of calling.....	<b>errnoGet()</b> 571
	task. set	error status value of calling.....	<b>errnoSet()</b> 573
	specified task. get	error status value of.....	<b>errnoOfTaskGet()</b> 572
	specified task. set	error status value of.....	<b>errnoOfTaskSet()</b> 572
	formatted string to standard	error stream. write.....	<b>printErr()</b> 996
	map error number to	error string (ANSI).....	<b>strerror()</b> 1274
	map error number to	error string (POSIX).....	<b>strerror_r()</b> 1274
	/contents of register edi (also	esi - eax) (x86/SimNT).....	<b>edi()</b> 568
	record. delete	Ethernet multicast address.....	<b>etherMultiDel()</b> 574
	library to handle	Ethernet multicast addresses.....	<b>etherMultiLib</b> 104
	handle controller-bus reset	event.....	<b>scsiMgrBusReset()</b> 1146
	manager of SCSI (controller)	event. notify SCSI.....	<b>scsiMgrEventNotify()</b> 1147
	trigger user-defined	event.....	<b>trgEvent()</b> 1397
	return ID of WindView	event buffer (WindView).....	<b>wvEvtBufferGet()</b> 1470
	file destination for	event data.....	<b>wvFileUploadPathLib</b> 350
	create file for depositing	event data (Windview).....	<b>fileUploadPathCreate()</b> 592
	include WDB user	event library.....	<b>wdbUserEvtLibInit()</b> 1463
	WDB user	event library.....	<b>wdbUserEvtLib</b> 348
	initialize	event log (WindView).....	<b>wvEvtLogInit()</b> 1472
	(WindView).	event logging control library.....	<b>wvLib</b> 351
	message queue. start	event notification process for.....	<b>msgQEvStart()</b> 859
	message queue. stop	event notification process for.....	<b>msgQEvStop()</b> 860
	semaphore. start	event notification process for.....	<b>semEvStart()</b> 1179
	semaphore. stop	event notification process for.....	<b>semEvStop()</b> 1181
	post user	event string to host tools.....	<b>wdbUserEvtPost()</b> 1463
	machine. send	event to SCSI controller state.....	<b>scsiMgrCtrlEvent()</b> 1146
	send	event to thread state machine.....	<b>scsiMgrThreadEvent()</b> 1148
	log user-defined	event (WindView).....	<b>wvEvent()</b> 1469
	(WindView). close	event-destination file.....	<b>fileUploadPathClose()</b> 592
	(WindView). write to	event-destination file.....	<b>fileUploadPathWrite()</b> 593
	create	event-log header (WindView).....	<b>wvLogHeaderCreate()</b> 1474
	set or display	eventpoints (WindView).....	<b>e()</b> 567
	remove address filter for	events.....	<b>wvNetAddressFilterClear()</b> 1475
	specify address filter for	events.....	<b>wvNetAddressFilterSet()</b> 1475
	deactivate specific network	events.....	<b>wvNetEventDisable()</b> 1477
	activate specific network	events.....	<b>wvNetEventEnable()</b> 1478
	remove port number filter for	events.....	<b>wvNetPortFilterClear()</b> 1480
	specify address filter for	events.....	<b>wvNetPortFilterSet()</b> 1481

	Keyword	Name	Page
	wait for event(s) .....	<b>eventReceive()</b>	575
	send event(s) .....	<b>eventSend()</b>	577
	trigger events control library .....	<b>trgLib</b>	324
	clear all events for current task .....	<b>eventClear()</b>	575
(WindView).	clear class of events from those being logged .....	<b>wvEvtClassClear()</b>	1470
	clear all classes of events from those logged/ .....	<b>wvEvtClassClearAll()</b>	1470
	VxWorks events library .....	<b>eventLib</b>	104
queues. VxWorks	events support for message .....	<b>msgQEvLib</b>	183
	VxWorks events support for semaphores .....	<b>semEvLib</b>	265
	start logging events to buffer (WindView) .....	<b>wvEvtLogStart()</b>	1472
	stop logging events to buffer (WindView) .....	<b>wvEvtLogStop()</b>	1473
	start upload of events to host (WindView) .....	<b>wvUploadStart()</b>	1486
	stop upload of events to host (WindView) .....	<b>wvUploadStop()</b>	1487
	set class of events to log (WindView) .....	<b>wvEvtClassSet()</b>	1471
end reporting of network	events to WindView .....	<b>wvNetDisable()</b>	1476
begin reporting network	events to WindView .....	<b>wvNetEnable()</b>	1476
instrument VxWorks	Events (WindView) .....	<b>wvEventInst()</b>	1469
extra copy of task name	events (WindView). preserve .....	<b>wvTaskNamesPreserve()</b>	1484
upload preserved task name	events (WindView) .....	<b>wvTaskNamesUpload()</b>	1485
level. enable network	events with specific priority .....	<b>wvNetLevelAdd()</b>	1479
level. disable network	events with specific priority .....	<b>wvNetLevelRemove()</b>	1479
generic	exception handling facilities .....	<b>excLib</b>	106
initialize	exception handling package .....	<b>excInit()</b>	580
connect C routine to critical	exception vector (PowerPC/ .....	<b>excCrtConnect()</b>	578
connect C routine to	exception vector (PowerPC) .....	<b>excConnect()</b>	577
/C routine to asynchronous	exception vector (PowerPC,/ .....	<b>excIntConnect()</b>	580
ARM). get CPU	exception vector (PowerPC, .....	<b>excVecGet()</b>	582
ARM). set CPU	exception vector (PowerPC, .....	<b>excVecSet()</b>	584
x86, ARM,/ write-protect	exception vector table (68K, .....	<b>intVecTableWriteProtect()</b>	704
architecture-specific	exception-handling facilities .....	<b>excArchLib</b>	105
initialize	exception/interrupt vectors .....	<b>excVecInit()</b>	582
routine to be called with	exceptions. specify .....	<b>excHookAdd()</b>	579
handle task-level	exceptions .....	<b>excTask()</b>	582
add interface to RIP	exclusion list .....	<b>ripIfExcludeListAdd()</b>	1093
delete interface from RIP	exclusion list .....	<b>ripIfExcludeListDelete()</b>	1094
show RIP interface	exclusion list .....	<b>ripIfExcludeListShow()</b>	1094
compare-and-exchange/	execute atomic .....	<b>pentiumBtc()</b>	951
compare-and-exchange/	execute atomic .....	<b>pentiumBts()</b>	951
instruction CPUID.	execute serializing .....	<b>pentiumSerialize()</b>	973
remote machine.	execute shell command on .....	<b>rcmd()</b>	1057
delay task from	executing .....	<b>taskDelay()</b>	1327
functions. time single	execution of function or .....	<b>timex()</b>	1385
cancel	execution of thread (POSIX) .....	<b>pthread_cancel()</b>	1018
shell	execution routines .....	<b>shellLib</b>	275
display synopsis of	execution timer facilities .....	<b>timexHelp()</b>	1387
	execution timer facilities .....	<b>timexLib</b>	323
include	execution timer library .....	<b>timexInit()</b>	1388
group of/ time repeated	executions of function or .....	<b>timexN()</b>	1388
	exit task (ANSI) .....	<b>exit()</b>	584
from DNS packet.	expand DNS compressed name .....	<b>resolvDNExpand()</b>	1074

	Keyword	Name	Page
type and value of call's next/	expand to expression having.....	<b>va_arg()</b>	1428
compute	exponential value (ANSI).....	<b>exp()</b>	585
compute	exponential value (ANSI).....	<b>expf()</b>	585
value of call's/ expand to	expression having type and.....	<b>va_arg()</b>	1428
call allocation	failure handler (C++).....	<b>cplusCallNewHandler()</b>	502
with variable argument list to	fd. write string formatted.....	<b>vfprintf()</b>	1430
write formatted string to	fd.....	<b>fprintf()</b>	588
add logging	fd.....	<b>logFdAdd()</b>	749
delete logging	fd.....	<b>logFdDelete()</b>	750
set primary logging	fd.....	<b>logFdSet()</b>	750
value. validate open	fd and return driver-specific.....	<b>iosFdValue()</b>	711
input/output/error. get	fd for global standard.....	<b>ioGlobalStdGet()</b>	706
input/output/error. set	fd for global standard.....	<b>ioGlobalStdSet()</b>	707
return	fd for stream (POSIX).....	<b>fileno()</b>	591
input/output/error. get	fd for task standard.....	<b>ioTaskStdGet()</b>	713
input/output/error. set	fd for task standard.....	<b>ioTaskStdSet()</b>	713
display list of	fd names in system.....	<b>iosFdShow()</b>	711
open file specified by	fd (POSIX).....	<b>fdopen()</b>	587
disk. create	FDISK-like partition table on.....	<b>usrFdiskPartCreate()</b>	1421
read	FDISK-style partition handler.....	<b>usrFdiskPartLib</b>	337
pend on set of	FDISK-style partition table.....	<b>usrFdiskPartRead()</b>	1422
fds.	.....	<b>select()</b>	1169
default input/output/error	fds. set shell's.....	<b>shellOrigStdSet()</b>	1214
backplane address from device	field. extract.....	<b>bootBpAnchorExtract()</b>	432
extract net mask	field from Internet address.....	<b>bootNetmaskExtract()</b>	434
populate rcvAddr	fields for ifRcvAddressTable.....	<b>rcvEtherAddrGet()</b>	1058
partition. initialize	fields in SCSI logical.....	<b>scsiBlkDevInit()</b>	1138
change name of	file.....	<b>rename()</b>	1072
remove	file.....	<b>rm()</b>	1107
update time on	file.....	<b>utime()</b>	1427
write bytes to	file.....	<b>write()</b>	1468
close	file.....	<b>close()</b>	496
create	file.....	<b>creat()</b>	508
read string from	file.....	<b>fioRdString()</b>	595
open	file.....	<b>open()</b>	945
remove	file (ANSI).....	<b>remove()</b>	1071
indicator to beginning of	file (ANSI). /file position.....	<b>rewind()</b>	1082
modify MS-DOS	file attributes of many files.....	<b>xattrib()</b>	1489
directory. modify MS-DOS	file attributes on file or.....	<b>attrib()</b>	423
data.	file destination for event.....	<b>wvFileUploadPathLib</b>	350
create remote	file device.....	<b>netDevCreate()</b>	911
size. create remote	file device with fixed buffer.....	<b>netDevCreate2()</b>	912
install network remote	file driver.....	<b>netDrv()</b>	913
install applet to test if	file exists.....	<b>netDrvFileDoesNotExistInstall()</b>	914
(Windview). create	file for depositing event data.....	<b>fileUploadPathCreate()</b>	592
helper	file for igmp Mib.....	<b>m2IgmP</b>	141
TrueFFS configuration	file for VxWorks.....	<b>tffsConfig</b>	315
get	file from remote system.....	<b>tftpGet()</b>	1372
mv	file into other directory.....	<b>mv()</b>	905
file/directory. copy	file into other.....	<b>cp()</b>	502

	Keyword	Name	Page
network remote	file I/O driver. ....	<b>netDrv</b>	190
device. write	file marks to SCSI sequential.....	<b>scsiWrtFileMarks()</b>	1167
find module by	file name and path. ....	<b>moduleFindByNameAndPath()</b>	832
generate temporary	file name (ANSI). ....	<b>tmpnam()</b>	1391
/object module by specifying	file name or module ID. ....	<b>unld()</b>	1416
standard input/output/error	FILE of current task. return .....	<b>stdioFp()</b>	1269
read bytes from	file or device.....	<b>read()</b>	1064
MS-DOS file attributes on	file or directory. modify.....	<b>attrib()</b>	423
test error indicator for	file pointer (ANSI). ....	<b>ferror()</b>	589
display	file pointer internals. ....	<b>stdioShow()</b>	1270
stream/ store current value of	file position indicator for .....	<b>fgetpos()</b>	590
stream (ANSI). set	file position indicator for .....	<b>fseek()</b>	618
stream (ANSI). set	file position indicator for .....	<b>fsetpos()</b>	619
return current value of	file position indicator for/ .....	<b>ftell()</b>	620
beginning of file (ANSI). set	file position indicator to .....	<b>rewind()</b>	1082
delete	file (POSIX). ....	<b>unlink()</b>	1418
truncate	file (POSIX). ....	<b>ftruncate()</b>	633
set	file read/write pointer.....	<b>lseek()</b>	760
open	file specified by fd (POSIX).....	<b>fdopen()</b>	587
open	file specified by name (ANSI). ....	<b>fopen()</b>	599
open	file specified by name (ANSI). ....	<b>freopen()</b>	612
(POSIX). get	file status information .....	<b>fstat()</b>	619
(POSIX). get	file status information .....	<b>fstatfs()</b>	620
pathname (POSIX). get	file status information using .....	<b>stat()</b>	1268
pathname (POSIX). get	file status information using .....	<b>statfs()</b>	1268
device and create rt11Fs	file system. initialize .....	<b>rt11FsMkfs()</b>	1127
consistency checking on MS-DOS	file system. perform.....	<b>chkdsk()</b>	492
mount NFS	file system. ....	<b>nfsMount()</b>	941
set rt11Fs	file system date.....	<b>rt11FsDateSet()</b>	1125
create	file system device. ....	<b>dosFsDevCreate()</b>	559
MS-DOS media-compatible	file system formatting/ .....	<b>dosFsFmtLib</b>	86
or CDROM. mount DOS	file system from ATA hard disk.....	<b>usrAtaConfig()</b>	1419
mount DOS	file system from floppy disk.....	<b>usrFdConfig()</b>	1420
disk. mount DOS	file system from IDE hard.....	<b>usrIdeConfig()</b>	1424
exported file systems. remove	file system from list of.....	<b>nfsUnexport()</b>	942
associate device with ntPassFs	file system functions. ....	<b>ntPassFsDevInit()</b>	943
associate device with passFs	file system functions. ....	<b>passFsDevInit()</b>	949
pass-through (to Windows NT)	file system library. ....	<b>ntPassFsLib</b>	198
raw block device	file system library. ....	<b>rawFsLib</b>	226
RT-11 media-compatible	file system library. ....	<b>rt11FsLib</b>	239
tape sequential device	file system library. ....	<b>tapeFsLib</b>	302
ISO 9660 CD-ROM read-only	file system library. ....	<b>cdromFsLib</b>	59
MS-DOS media-compatible	file system library. ....	<b>dosFsLib</b>	86
pass-through (to UNIX)	file system library (VxSim). ....	<b>passFsLib</b>	200
Network	File System (NFS) I/O driver. ....	<b>nfsDrv</b>	195
Network	File System (NFS) library. ....	<b>nfsLib</b>	197
library. Network	File System (NFS) server.....	<b>nfsdLib</b>	193
initialize	file system on block device. ....	<b>diskInit()</b>	544
exported. specify	file system to be NFS .....	<b>nfsExport()</b>	939
subroutine library.	file system user interface.....	<b>usrFsLib</b>	339

	Keyword	Name	Page
system from list of exported	file systems. remove file .....	nfsUnexport()	942
specified host. mount all	file systems exported by .....	nfsMountAll()	942
display exported	file systems of remote host.....	nfsExportShow()	939
put	file to remote system.....	tftpPut()	1374
library.	File Transfer Protocol (FTP) .....	ftpLib	115
server.	File Transfer Protocol (FTP) .....	ftpdLib	113
library. Trivial	File Transfer Protocol server .....	tftpdLib	318
client library. Trivial	File Transfer Protocol (TFTP).....	tftpLib	319
POSIX	file truncation.....	ftruncate	117
create temporary binary	file (Unimplemented) (ANSI).....	tmpfile()	1391
transfer	file via TFTP.....	tftpCopy()	1368
interface. transfer	file via TFTP using stream .....	tftpXfer()	1376
close event-destination	file (WindView).....	fileUploadPathClose()	592
write to event-destination	file (WindView).....	fileUploadPathWrite()	593
format. archive named	file/dir onto tape in tar.....	tarArchive()	1322
copy file into other	file/directory.....	cp()	502
MS-DOS file attributes of many	files. modify.....	xattrib()	1489
memory device for multiple	files. create.....	memDevCreateDir()	810
extract all	files from tar formatted tape.....	tarExtract()	1323
copy hierarchy of	files with wildcards.....	xcopy()	1490
delete hierarchy of	files with wildcards.....	xdelete()	1490
library. Berkeley Packet	Filter (BPF) I/O driver.....	bpfDrv	35
create Berkeley Packet	Filter device.....	bpfDevCreate()	442
destroy Berkeley Packet	Filter device.....	bpfDevDelete()	442
initialize IP	filter facility.....	ipFilterLibInit()	716
remove address	filter for events.....	wvNetAddressFilterClear()	1475
specify address	filter for events.....	wvNetAddressFilterSet()	1475
remove port number	filter for events.....	wvNetPortFilterClear()	1480
specify address	filter for events.....	wvNetPortFilterSet()	1481
remove update	filter from RIP interface.....	ripSendHookDelete()	1104
delete IP	filter hook routine.....	ipFilterHookDelete()	716
IP	filter hooks library.....	ipFilterLib	128
add update	filter to RIP interface.....	ripSendHookAdd()	1103
/distributed name database	filtered by type (VxFusion).....	distNameFilterShow()	552
prevent strict border gateway	filtering.....	ripFilterDisable()	1092
activate strict border gateway	filtering.....	ripFilterEnable()	1093
create remote file device with	fixed buffer size.....	netDevCreate2()	912
change network interface	flags.....	ifFlagChange()	663
get network interface	flags.....	ifFlagGet()	664
ID. get	flags associated with module .....	moduleFlagsGet()	832
specify	flags for network interface.....	ifFlagSet()	664
clear end-of-file and error	flags for stream (ANSI).....	clearerr()	493
low level I/O access to	flash components.....	tffsRawio()	1366
write to boot-image region of	flash device.....	tffsBootImagePut()	1362
TrueFFS. format	flash device for use with.....	tffsDevFormat()	1364
return very large	float.....	infinityf()	682
probe for presence of	floating-point coprocessor.....	fppProbe()	601
context. restore	floating-point coprocessor .....	fppRestore()	601
context. save	floating-point coprocessor .....	fppSave()	603
architecture-dependent	floating-point coprocessor/ .....	fppArchLib	109

	Keyword	Name	Page
	support. initialize floating-point coprocessor .....	<b>fppInit()</b>	600
	support library. floating-point coprocessor .....	<b>fppLib</b>	112
	library. high-level floating-point emulation .....	<b>mathSoftLib</b>	155
	scanning library. floating-point formatting and .....	<b>floatLib</b>	109
	initialize floating-point I/O support .....	<b>floatInit()</b>	596
	hardware floating-point math library .....	<b>mathHardLib</b>	155
	initialize hardware floating-point math support .....	<b>mathHardInit()</b>	803
	initialize software floating-point math support .....	<b>mathSoftInit()</b>	803
	integer and fraction/ separate floating-point number into .....	<b>modf()</b>	827
	normalized fraction and/ break floating-point number into .....	<b>frexp()</b>	613
	print contents of task's floating-point registers .....	<b>fppTaskRegsShow()</b>	606
	task TCB. get floating-point registers from .....	<b>fppTaskRegsGet()</b>	605
	task. set floating-point registers of .....	<b>fppTaskRegsSet()</b>	605
	initialize floating-point show facility .....	<b>fppShowInit()</b>	604
	floating-point show routines .....	<b>fppShow</b>	113
	mount DOS file system from floppy disk .....	<b>usrFdConfig()</b>	1420
	floppy disk initialization .....	<b>usrFd</b>	336
	convert format string .....	<b>fioFormatV()</b>	594
	log formatted error message .....	<b>logMsg()</b>	757
	formatted I/O library .....	<b>fioLib</b>	108
	initialize formatted I/O support library .....	<b>fioLibInit()</b>	595
	convert broken-down time into formatted string (ANSI) .....	<b>strftime()</b>	1275
	(ANSI). write formatted string to buffer .....	<b>sprintf()</b>	1256
	write formatted string to fd .....	<b>fdprintf()</b>	588
	error stream. write formatted string to standard .....	<b>printErr()</b>	996
	output stream (ANSI). write formatted string to standard .....	<b>printf()</b>	997
	(ANSI). write formatted string to stream .....	<b>vfprintf()</b>	1431
	(ANSI). write formatted string to stream .....	<b>fprintf()</b>	606
	extract all files from tar formatted tape .....	<b>tarExtract()</b>	1323
	display all contents of tar formatted tape .....	<b>tarToc()</b>	1324
	argument list to/ write string formatted with variable .....	<b>vsprintf()</b>	1446
	argument list to/ write string formatted with variable .....	<b>vfdprintf()</b>	1430
	argument list to/ write string formatted with variable .....	<b>vprintf()</b>	1446
	library. floating-point formatting and scanning .....	<b>floatLib</b>	109
	media-compatible file system formatting library. MS-DOS .....	<b>dosFsFmtLib</b>	86
	device. issue FORMAT_UNIT command to SCSI .....	<b>scsiFormatUnit()</b>	1142
	ports enabled for broadcast forwarding. show .....	<b>proxyPortShow()</b>	1004
	port. disable broadcast forwarding for particular .....	<b>proxyPortFwdOff()</b>	1003
	port. enable broadcast forwarding for particular .....	<b>proxyPortFwdOn()</b>	1004
	/number into normalized fraction and power of 2/ .....	<b>frexp()</b>	613
	/number into integer and fraction parts (ANSI) .....	<b>modf()</b>	827
	form frame with link-layer address .....	<b>muxAddressForm()</b>	870
	list directory contents via FTP .....	<b>ftpLs()</b>	627
	initiate transfer via FTP .....	<b>ftpXfer()</b>	631
	RFC reply code. send FTP command and get complete .....	<b>ftpCommandEnhanced()</b>	622
	send FTP command and get reply .....	<b>ftpCommand()</b>	621
	get FTP command reply .....	<b>ftpReplyGet()</b>	628
	get FTP command reply .....	<b>ftpReplyGetEnhanced()</b>	629
	get completed FTP data connection .....	<b>ftpDataConnGet()</b>	623
	PASV mode. initialize FTP data connection using .....	<b>ftpDataConnInitPassiveMode()</b>	624

	Keyword	Name	Page
	mode. initialize	FTP data connection using PORT .....	<b>ftpDataConnInit()</b> 623
	File Transfer Protocol	(FTP) library. ....	<b>ftpLib</b> 115
	set debug level of	ftp library routines. ....	<b>ftpLibDebugOptionSet()</b> 626
	File Transfer Protocol	(FTP) server. ....	<b>ftpLib</b> 113
	log in to remote	FTP server. ....	<b>ftpLogin()</b> 627
	get control connection to	FTP server on specified host. ....	<b>ftpHookup()</b> 626
	terminate	FTP server task. ....	<b>ftpdDelete()</b> 625
	initialize	FTP server task. ....	<b>ftpdInit()</b> 625
	set applette to stop	FTP transient host responses. ....	<b>ftpTransientFatalInstall()</b> 631
	get parameters for host	FTP_TRANSIENT responses. ....	<b>ftpTransientConfigGet()</b> 630
	set parameters for host	FTP_TRANSIENT responses. ....	<b>ftpTransientConfigSet()</b> 630
	initialize	Fujitsu MB86930 cache library. ....	<b>cacheMb930LibInit()</b> 459
	manager.	full-featured memory partition .....	<b>memLib</b> 158
	gate type(int/trap), and	gate selector (x86). /vector, .....	<b>intVecGet2()</b> 699
	selector/ get CPU vector,	gate type(int/trap), and gate .....	<b>intVecGet2()</b> 699
	selector/ set CPU vector,	gate type(int/trap), and .....	<b>intVecSet2()</b> 703
	prevent strict border	gateway filtering. ....	<b>ripFilterDisable()</b> 1092
	activate strict border	gateway filtering. ....	<b>ripFilterEnable()</b> 1093
		general semaphore library. ....	<b>semLib</b> 266
	unlock	(give) semaphore (POSIX). ....	<b>sem_post()</b> 1193
	address (VxMP). convert	global address to local .....	<b>smObjGlobalToLocal()</b> 1243
	convert local address to	global address (VxMP). ....	<b>smObjLocalToGlobal()</b> 1245
	get MIB-II RIP-group	global counters. ....	<b>m2RipGlobalCountersGet()</b> 793
	Register/ get content of	Global Descriptor Table. ....	<b>vxGdtrGet()</b> 1452
	initialize	global mapping. ....	<b>vmBaseGlobalMapInit()</b> 1431
	initialize	global mapping (VxVMI). ....	<b>vmGlobalMapInit()</b> 1439
	get fd for	global standard/ .....	<b>ioGlobalStdGet()</b> 706
	set fd for	global standard/ .....	<b>ioGlobalStdSet()</b> 707
	initialize	global state for MUX. ....	<b>muxLibInit()</b> 881
	get MIB-II ICMP-group	global variables. ....	<b>m2IcmpGroupInfoGet()</b> 771
	/to virtual space in shared	global virtual mem (VxVMI). ....	<b>vmGlobalMap()</b> 1438
	information (VxVMI). get	global virtual memory .....	<b>vmGlobalInfoGet()</b> 1437
	environment/ perform non-local	goto by restoring saved .....	<b>longjmp()</b> 759
	compute smallest integer	greater than or equal to/ .....	<b>ceil()</b> 489
	compute smallest integer	greater than or equal to/ .....	<b>ceilf()</b> 490
	Packet InterNet	Groper (PING) library. ....	<b>pingLib</b> 209
	mount DOS file system from IDE	hard disk. ....	<b>usrIdeConfig()</b> 1424
	initialize	hard disk driver. ....	<b>usrAtaInit()</b> 1420
	mount DOS file system from ATA	hard disk or CDROM. ....	<b>usrAtaConfig()</b> 1419
	initialize system	hardware. ....	<b>sysHwInit()</b> 1304
	Internet address. resolve	hardware address for specified .....	<b>arpResolve()</b> 414
	set	hardware breakpoint. ....	<b>bh()</b> 429
	library.	hardware floating-point math .....	<b>mathHardLib</b> 155
	support. initialize	hardware floating-point math .....	<b>mathHardInit()</b> 803
	connect C routine to	hardware interrupt. ....	<b>intConnect()</b> 683
	disabled. inform SCSI that	hardware snooping of caches is .....	<b>scsiCacheSnoopDisable()</b> 1140
	enabled. inform SCSI that	hardware snooping of caches is .....	<b>scsiCacheSnoopEnable()</b> 1140
	remove	hash node from hash table. ....	<b>hashTblRemove()</b> 650
	table. put	hash node into specified hash .....	<b>hashTblPut()</b> 649
	specified key. find	hash node that matches .....	<b>hashTblFind()</b> 648

	Keyword	Name	Page
	create hash table.	<b>hashTblCreate()</b>	646
	delete hash table.	<b>hashTblDelete()</b>	647
	destroy hash table.	<b>hashTblDestroy()</b>	647
call routine for each node in	hash table.	<b>hashTblEach()</b>	648
	initialize hash table.	<b>hashTblInit()</b>	649
put hash node into specified	hash table.	<b>hashTblPut()</b>	649
remove hash node from	hash table.	<b>hashTblRemove()</b>	650
terminate	hash table.	<b>hashTblTerminate()</b>	650
test	hash table integrity.	<b>dcacheHashTest()</b>	518
initialize	hash table library.	<b>hashLibInit()</b>	646
multiplicative	hashing function.	<b>hashFuncMultiply()</b>	644
iterative scaling	hashing function for strings.	<b>hashFuncIterScale()</b>	643
remainder technique.	hashing function using.	<b>hashFuncModulo()</b>	644
transfer log	header to host (WindView).	<b>wvLogHeaderUpload()</b>	1474
attach link-level	header to packet.	<b>muxLinkHeaderCreate()</b>	881
create event-log	header (WindView).	<b>wvLogHeaderCreate()</b>	1474
kernel	heap version of <b>netPoolInit()</b> .	<b>netPoolKheapInit()</b>	927
display task monitoring	help menu.	<b>spyHelp()</b>	1258
display debugging	help menu.	<b>dbgHelp()</b>	513
display NFS	help menu.	<b>nfsHelp()</b>	940
	helper file for igmp Mib.	<b>m2Igmplib</b>	141
test whether character is	hexadecimal digit (ANSI).	<b>isxdigit()</b>	725
wildcards. copy	hierarchy of files with.	<b>xcopy()</b>	1490
wildcards. delete	hierarchy of files with.	<b>xdelete()</b>	1490
display or set size of shell	history.	<b>shellHistory()</b>	1212
display or set size of shell	history.	<b>h()</b>	643
management library.	Hitachi SH7040 cache.	<b>cacheSh7040Lib</b>	51
management library.	Hitachi SH7604/SH7615 cache.	<b>cacheSh7604Lib</b>	51
management library.	Hitachi SH7700 cache.	<b>cacheSh7700Lib</b>	52
library.	Hitachi SH7700 MMU support.	<b>mmuSh7700Lib</b>	168
management library.	Hitachi SH7729 cache.	<b>cacheSh7729Lib</b>	53
management library.	Hitachi SH7750 cache.	<b>cacheSh7750Lib</b>	53
library.	Hitachi SH7750 MMU support.	<b>mmuSh7750Lib</b>	172
sample authentication	hook.	<b>ripAuthHook()</b>	1084
remove route	hook.	<b>ripRouteHookDelete()</b>	1102
initialize task	hook facilities.	<b>taskHookInit()</b>	1330
permanent address storage	hook for server. assign.	<b>dhcpsAddressHookAdd()</b>	538
assign permanent lease storage	hook for server.	<b>dhcpsLeaseHookAdd()</b>	541
remove authentication	hook from RIP interface.	<b>ripAuthHookDelete()</b>	1087
remove table bypass	hook from RIP interface.	<b>ripLeakHookDelete()</b>	1097
PPP	hook library.	<b>pppHookLib</b>	212
task	hook library.	<b>taskHookLib</b>	307
delete IP filter	hook routine.	<b>ipFilterHookDelete()</b>	716
previously added module create	hook routine. delete.	<b>moduleCreateHookDelete()</b>	829
add	hook routine on unit basis.	<b>pppHookAdd()</b>	983
delete	hook routine on unit basis.	<b>pppHookDelete()</b>	983
initialize task	hook show facility.	<b>taskHookShowInit()</b>	1331
task	hook show routines.	<b>taskHookShow</b>	308
routing tables. add	hook to bypass RIP and kernel.	<b>ripLeakHookAdd()</b>	1096
non-RIP routes into RIP. add	hook to install static and.	<b>ripRouteHookAdd()</b>	1099

	Keyword	Name	Page
add authentication	hook to RIP interface.....	<b>ripAuthHookAdd()</b>	1085
IP filter	hooks library.....	<b>ipFilterLib</b>	128
specify network interface	hop count.....	<b>ifMetricSet()</b>	669
DNS server for IP address of	host. query.....	<b>resolvGetHostByName()</b>	1076
log in to remote	host.....	<b>rlogin()</b>	1105
to FTP server on specified	host. get control connection.....	<b>ftpHookup()</b>	626
file systems of remote	host. display exported.....	<b>nfsExportShow()</b>	939
systems exported by specified	host. mount all file.....	<b>nfsMountAll()</b>	942
(DHCP) run-time/ Dynamic	Host Configuration Protocol.....	<b>dhcpcLib</b>	73
(DHCP) server/ Dynamic	Host Configuration Protocol.....	<b>dhcpsLib</b>	76
TSFS. target	host connection library using.....	<b>wvTsfsUploadPathLib</b>	358
delete	host from host table.....	<b>hostDelete()</b>	652
get parameters for	host FTP_TRANSIENT responses.....	<b>ftpTransientConfigGet()</b>	630
set parameters for	host FTP_TRANSIENT responses.....	<b>ftpTransientConfigSet()</b>	630
address. look up	host in host table by Internet.....	<b>hostGetByAddr()</b>	653
look up	host in host table by name.....	<b>hostGetByName()</b>	654
test that remote	host is reachable.....	<b>ping()</b>	977
query DNS server for	host name of IP address.....	<b>resolvGetHostByAddr()</b>	1075
applette to stop FTP transient	host responses. set.....	<b>ftpTransientFatalInstall()</b>	631
add	host to host table.....	<b>hostAdd()</b>	652
post user event string to	host tools.....	<b>wdbUserEvtPost()</b>	1463
establish upload path to	host using socket (Windview).....	<b>sockUploadPathCreate()</b>	1254
open upload path to	host using TSFS socket/.....	<b>tsfsUploadPathCreate()</b>	1402
transfer log header to	host (WindView).....	<b>wvLogHeaderUpload()</b>	1474
start upload of events to	host (WindView).....	<b>wvUploadStart()</b>	1486
stop upload of events to	host (WindView).....	<b>wvUploadStop()</b>	1487
address. get local address	((host number)) from Internet.....	<b>inet_lnaof()</b>	676
address from network and	((host number))s. form Internet.....	<b>inet_makeaddr()</b>	676
address from network and	((host number))s. form Internet.....	<b>inet_makeaddr_b()</b>	677
add host to	host table.....	<b>hostAdd()</b>	652
delete host from	host table.....	<b>hostDelete()</b>	652
display	host table.....	<b>hostShow()</b>	654
initialize network	host table.....	<b>hostTblInit()</b>	655
address. look up host in	host table by Internet.....	<b>hostGetByAddr()</b>	653
look up host in	host table by name.....	<b>hostGetByName()</b>	654
synchronization. initialize	host table subroutine library.....	<b>hostLib</b>	118
synchronization.	host/target symbol table.....	<b>symSyncLibInit()</b>	1295
for Solaris and VxSim for	host/target symbol table.....	<b>symSyncLib</b>	297
compute	HP). /disk driver (VxSim.....	<b>unixDrv</b>	332
compute	hyperbolic cosine (ANSI).....	<b>cosh()</b>	501
compute	hyperbolic cosine (ANSI).....	<b>coshf()</b>	501
compute	hyperbolic sine (ANSI).....	<b>sinh()</b>	1230
compute	hyperbolic sine (ANSI).....	<b>sinhf()</b>	1230
compute	hyperbolic tangent (ANSI).....	<b>tanh()</b>	1319
compute	hyperbolic tangent (ANSI).....	<b>tanhf()</b>	1319
display statistics for	ICMP.....	<b>icmpstatShow()</b>	658
all resources used to access	ICMP group. delete.....	<b>m2IcmpDelete()</b>	771
routines.	ICMP Information display.....	<b>icmpShow</b>	119
function. implement	ICMP router discovery control.....	<b>rdCtl()</b>	1059
function. implement	ICMP router discovery.....	<b>rdisc()</b>	1061

	Keyword	Name	Page
function. initialize	ICMP router discovery.....	<b>rdiscInit()</b>	1062
library.	ICMP router discovery server.....	<b>rdiscLib</b>	230
initialize	ICMP show routines.....	<b>icmpShowInit()</b>	657
initialize MIB-II	ICMP-group access.....	<b>m2IcmpInit()</b>	772
Agents. MIB-II	ICMP-group API for SNMP.....	<b>m2IcmpLib</b>	139
get MIB-II	ICMP-group global variables.....	<b>m2IcmpGroupInfoGet()</b>	771
mount DOS file system from	IDE hard disk.....	<b>usrIdeConfig()</b>	1424
	IDE initialization.....	<b>usrIde</b>	340
compare keys as 32 bit	identifiers.....	<b>hashKeyCmp()</b>	645
display current remote	identity.....	<b>whoami()</b>	1468
populate rcvAddr fields for	ifRcvAddressTable.....	<b>rcvEtherAddrGet()</b>	1058
layers above. test if	ifStackTable interface has no.....	<b>stackEntryIsTop()</b>	1267
insert or remove entry in	ifTable.....	<b>m2IfTableUpdate()</b>	783
resolution function for	ifType/protocol. get address.....	<b>muxAddrResFuncGet()</b>	873
display statistics for	IGMP.....	<b>igmpstatShow()</b>	674
routines.	IGMP information display.....	<b>igmpShow</b>	121
helper file for	igmp Mib.....	<b>m2Igmplib</b>	141
initialize	IGMP show routines.....	<b>igmpShowInit()</b>	673
return unique interface	index.....	<b>ifIndexAlloc()</b>	665
interface name given interface	index. returns.....	<b>ifIndexToIfName()</b>	667
returns interface	index given interface name.....	<b>ifNameToIfIndex()</b>	670
returns true if	index has been allocated.....	<b>ifIndexTest()</b>	667
interface	index library.....	<b>ifIndexLib</b>	119
(ANSI). test error	indicator for file pointer.....	<b>ferror()</b>	589
test end-of-file	indicator for stream (ANSI).....	<b>feof()</b>	588
/current value of file position	indicator for stream (ANSI).....	<b>fgetpos()</b>	590
set file position	indicator for stream (ANSI).....	<b>fseek()</b>	618
set file position	indicator for stream (ANSI).....	<b>fsetpos()</b>	619
/current value of file position	indicator for stream (ANSI).....	<b>ftell()</b>	620
(ANSI). set file position	indicator to beginning of file.....	<b>rewind()</b>	1082
domain zlib functions.	inflate code using public.....	<b>inflateLib</b>	123
	inflate compressed code.....	<b>inflate()</b>	682
thread attribute object/ set	inheritsched attribute in.....	<b>pthread_attr_setinheritsched()</b>	1014
thread/ get value of	inheritsched attribute in.....	<b>pthread_attr_getinheritsched()</b>	1008
double (ANSI). convert	initial portion of string to.....	<b>strtod()</b>	1281
activate task that has been	initialized.....	<b>taskActivate()</b>	1325
semaphore (POSIX).	initialize/open named.....	<b>sem_open()</b>	1191
(POSIX).	initiate asynchronous read.....	<b>aio_read()</b>	408
(POSIX).	initiate asynchronous write.....	<b>aio_write()</b>	410
	initiate connection to socket.....	<b>connect()</b>	497
I/O requests (POSIX).	initiate list of asynchronous.....	<b>lio_listio()</b>	735
negotiating transfer/	initiate or continue.....	<b>scsiSyncXferNegotiate()</b>	1162
negotiating wide parameters.	initiate or continue.....	<b>scsiWideXferNegotiate()</b>	1167
	initiate transfer via FTP.....	<b>ftpXfer()</b>	631
interrupt-level	input.....	<b>tyIRd()</b>	1410
characters from standard	input stream (ANSI). /convert.....	<b>scanf()</b>	1130
push character back into	input stream (ANSI).....	<b>ungetc()</b>	1413
next character from standard	input stream (ANSI). return.....	<b>getchar()</b>	635
read characters from standard	input stream (ANSI).....	<b>gets()</b>	638
get fd for global standard	input/output/error.....	<b>ioGlobalStdGet()</b>	706

	Keyword	Name	Page
set fd for global standard	input/output/error.....	<b>ioGlobalStdSet()</b>	707
get fd for task standard	input/output/error.....	<b>ioTaskStdGet()</b>	713
set fd for task standard	input/output/error.....	<b>ioTaskStdSet()</b>	713
set shell's default	input/output/error fds.....	<b>shellOrigStdSet()</b>	1214
current task. return standard	input/output/error FILE of .....	<b>stdioFp()</b>	1269
device. issue	INQUIRY command to SCSI.....	<b>scsiInquiry()</b>	1144
display specified number of	instructions. disassemble and.....	<b>l()</b>	729
automatic locking of kernel	instructions/data. /MB86930.....	<b>cacheMb930LockAuto()</b>	459
	instrument objects (WindView).....	<b>wvObjInst()</b>	1481
	instrument signals (WindView). .....	<b>wvSigInst()</b>	1483
(WindView).	instrument VxWorks Events.....	<b>wvEventInst()</b>	1469
(WindView). set object	instrumentation on/off .....	<b>wvObjInstModeSet()</b>	1482
convert string to	int (ANSI). .....	<b>atoi()</b>	422
round number to nearest	integer.....	<b>round()</b>	1114
round number to nearest	integer.....	<b>roundf()</b>	1114
truncate to	integer.....	<b>trunc()</b>	1400
truncate to	integer.....	<b>truncf()</b>	1401
Internet address to long	integer. convert dot notation .....	<b>inet_addr()</b>	675
double-precision value to	integer. convert .....	<b>rint()</b>	717
single-precision value to	integer. convert .....	<b>rintf()</b>	718
round number to nearest	integer.....	<b>iround()</b>	718
round number to nearest	integer.....	<b>iroundf()</b>	719
/floating-point number into	integer and fraction parts/ .....	<b>modf()</b>	827
convert string to long	integer (ANSI).....	<b>strtol()</b>	1284
string to unsigned long	integer (ANSI). convert .....	<b>strtoul()</b>	1285
compute absolute value of	integer (ANSI).....	<b>abs()</b>	404
generate pseudo-random	integer between 0 and RAND_MAX/ .....	<b>rand()</b>	1054
read next word (32-bit	integer) from stream. ....	<b>getw()</b>	640
to specified/ compute smallest	integer greater than or equal .....	<b>ceil()</b>	489
to specified/ compute smallest	integer greater than or equal .....	<b>ceilf()</b>	490
specified/ compute largest	integer less than or equal to.....	<b>floor()</b>	597
specified/ compute largest	integer less than or equal to.....	<b>floorf()</b>	597
write word (32-bit	integer) to stream. ....	<b>putw()</b>	1047
test hash table	integrity.....	<b>dcacheHashTest()</b>	518
make volume	integrity checking.....	<b>dosFsChkDsk()</b>	558
task. set	inter-cycle delay on polling .....	<b>muxTaskDelaySet()</b>	894
agents. MIB-II	interface-group API for SNMP.....	<b>m2IfLib</b>	139
initialize MIB-II	interface-group routines. ....	<b>m2IfInit()</b>	778
variables. get MIB-II	interface-group scalar .....	<b>m2IfGroupInfoGet()</b>	778
get MIB-II	interface-group table entry.....	<b>m2IfStackEntryGet()</b>	781
get MIB-II	interface-group table entry.....	<b>m2IfTblEntryGet()</b>	783
hardware address for specified	Internet address. resolve .....	<b>arpResolve()</b>	414
extract lease information from	Internet address.....	<b>bootLeaseExtract()</b>	433
extract net mask field from	Internet address.....	<b>bootNetmaskExtract()</b>	434
look up host in host table by	Internet address.....	<b>hostGetByAddr()</b>	653
address (((host number))) from	Internet address. get local .....	<b>inet_lnaof()</b>	676
return network number from	Internet address.....	<b>inet_netof()</b>	678
and ((host number))s. form	Internet address from network.....	<b>inet_makeaddr()</b>	676
and ((host number))s. form	Internet address from network.....	<b>inet_makeaddr_b()</b>	677
routines.	internet address manipulation .....	<b>inetLib</b>	121

	Keyword	Name	Page
	interface. get	Internet address of network .....	<b>ifAddrGet()</b> 659
	point-to-point peer. get	Internet address of .....	<b>ifDstAddrGet()</b> 662
	integer. convert dot notation	Internet address to long .....	<b>inet_addr()</b> 675
	library. Packet	InterNet Groper (PING) .....	<b>pingLib</b> 209
	string to address. convert	Internet network number from .....	<b>inet_network()</b> 679
	add routine to receive all	internet protocol packets. ....	<b>ipFilterHookAdd()</b> 715
	/all active connections for	Internet protocol sockets .....	<b>inetstatShow()</b> 681
	sessions. specify command	interpreter for telnet .....	<b>telnetdParserSet()</b> 1360
	routine to auxiliary clock	interrupt. connect .....	<b>sysAuxClkConnect()</b> 1297
	acknowledge bus	interrupt. ....	<b>sysBusIntAck()</b> 1300
	generate bus	interrupt. ....	<b>sysBusIntGen()</b> 1300
	routine to system clock	interrupt. connect .....	<b>sysClkConnect()</b> 1302
	connect routine to mailbox	interrupt. ....	<b>sysMailboxConnect()</b> 1306
	enable mailbox	interrupt. ....	<b>sysMailboxEnable()</b> 1307
	buffer after data store error	interrupt. clean up store .....	<b>cleanUpStoreBuffer()</b> 493
	connect C routine to hardware	interrupt. ....	<b>intConnect()</b> 683
	ARM). disable corresponding	interrupt bits (MIPS, PowerPC, .....	<b>intDisable()</b> 687
	ARM). enable corresponding	interrupt bits (MIPS, PowerPC, .....	<b>intEnable()</b> 688
	Register/ get content of	Interrupt Descriptor Table .....	<b>vxIdtrGet()</b> 1453
	routine (68K, x86,/ construct	interrupt handler for C .....	<b>intHandlerCreate()</b> 688
	routine (x86). construct	interrupt handler for C .....	<b>intHandlerCreateI86()</b> 689
	disable bus	interrupt level .....	<b>sysIntDisable()</b> 1305
	enable bus	interrupt level .....	<b>sysIntEnable()</b> 1305
	ARM, SimSolaris, SimNT/ set	interrupt level (68K, x86, .....	<b>intLevelSet()</b> 690
	architecture-dependent	interrupt library .....	<b>intArchLib</b> 123
	x86, ARM, SH,/ get current	interrupt lock-out level (68K, .....	<b>intLockLevelGet()</b> 693
	x86, ARM, SH,/ set current	interrupt lock-out level (68K, .....	<b>intLockLevelSet()</b> 693
	cancel	interrupt locks .....	<b>intUnlock()</b> 696
	and returns devices to	interrupt mode. /tMuxPollTask .....	<b>muxPollEnd()</b> 888
	get current	interrupt nesting depth. ....	<b>intCount()</b> 686
	/if current state is in	interrupt or task context. ....	<b>intContext()</b> 686
	user-defined system clock	interrupt routine .....	<b>usrClock()</b> 1420
	enable or disable	interrupt stack usage (x86). ....	<b>intStackEnable()</b> 695
	architecture-independent	interrupt subroutine library .....	<b>intLib</b> 125
	MIPS, SH, SimSolaris,/ get	interrupt vector (68K, x86, .....	<b>intVecGet()</b> 698
	connect C routine to critical	interrupt vector (PowerPC/ .....	<b>excIntCrtConnect()</b> 581
		interrupt-level input .....	<b>tyIRd()</b> 1410
		interrupt-level output .....	<b>tyITx()</b> 1410
	turn off auxiliary clock	interrupts .....	<b>sysAuxClkDisable()</b> 1297
	turn on auxiliary clock	interrupts .....	<b>sysAuxClkEnable()</b> 1298
	turn off system clock	interrupts .....	<b>sysClkDisable()</b> 1302
	turn on system clock	interrupts .....	<b>sysClkEnable()</b> 1303
	connect BSP serial device	interrupts .....	<b>sysSerialHwInit2()</b> 1316
	lock out	interrupts .....	<b>intLock()</b> 691
	/current task until time	interval elapses (POSIX). ....	<b>nanosleep()</b> 906
	buffer.	invert order of bytes in .....	<b>binvert()</b> 431
	components. low level	I/O access to flash .....	<b>tfFsRawio()</b> 1366
	initialize asynchronous	I/O (AIO) library .....	<b>aioPxLibInit()</b> 406
	asynchronous	I/O (AIO) library (POSIX) .....	<b>aioPxLib</b> 9
	asynchronous	I/O (AIO) show library .....	<b>aioPxShow</b> 13

	Keyword	Name	Page
	perform device-specific	I/O control function.....	scsiIoctl() 1145
	perform	I/O control function.....	ioctl() 704
	sequential access/	I/O control function for .....	scsiSeqIoctl() 1159
	find	I/O device in device list.....	iosDevFind() 709
	network remote file	I/O driver.....	netDrv 190
	Network File System (NFS)	I/O driver.....	nfsDrv 195
	pipe	I/O driver.....	pipeDrv 210
	install	I/O driver.....	iosDrvInstall() 710
	remove	I/O driver.....	iosDrvRemove() 710
	Berkeley Packet Filter (BPF)	I/O driver library.....	bpfDrv 35
		I/O interface library.....	ioLib 125
	formatted	I/O library.....	fioLib 108
	cached block	I/O library.....	cbioLib 55
	/error status of asynchronous	I/O operation (POSIX).....	aio_error() 408
	/return status of asynchronous	I/O operation (POSIX).....	aio_return() 409
	wait for asynchronous	I/O request(s) (POSIX).....	aio_suspend() 410
	initiate list of asynchronous	I/O requests (POSIX).....	lio_listio() 735
	initialize standard	I/O show facility.....	stdioShowInit() 1270
	initialize standard	I/O support.....	stdioInit() 1269
	initialize floating-point	I/O support.....	floatInit() 596
	initialize formatted	I/O support library.....	fioLibInit() 595
	add device to	I/O system.....	iosDevAdd() 708
	delete device from	I/O system.....	iosDevDelete() 708
	initialize	I/O system.....	iosInit() 712
	NFS driver. return	IO system driver number for.....	nfsDrvNumGet() 937
		I/O system library.....	iosLib 127
	initialize	I/O system show facility.....	iosShowInit() 712
		I/O system show routines.....	iosShow 127
	print synopsis of	I/O utility functions.....	ioHelp() 707
	perform	ioctl operation on device.....	cbioIoctl() 479
	DNS server for host name of	IP address. query.....	resolvGetHostByAddr() 1075
	query DNS server for	IP address of host.....	resolvGetHostByName() 1076
	initialize	IP filter facility.....	ipFilterLibInit() 716
	delete	IP filter hook routine.....	ipFilterHookDelete() 716
		IP filter hooks library.....	ipFilterLib 128
	all resources used to access	IP group. delete.....	m2IpDelete() 789
	get	IP MIB-II address entry.....	m2IpAddrTblEntryGet() 787
	bind socket to privileged	IP port.....	bindresvport() 431
	interface between BSD	IP protocol and MUX.....	ipProto 128
	information). display all	IP routes (summary).....	routeShow() 1120
	information). display all	IP routes (verbose).....	mRouteShow() 849
	traverse	IP routing table.....	routeTableWalk() 1123
	display	IP statistics.....	ipstatShow() 717
	initialize MIB-II	IP-group access.....	m2IpInit() 790
	MIB-II	IP-group API for SNMP agents.....	m2IpLib 142
	get MIB-II	IP-group scalar variables.....	m2IpGroupInfoGet() 789
	values. set MIB-II	IP-group variables to new.....	m2IpGroupInfoSet() 790
	system library.	ISO 9660 CD-ROM read-only file.....	cdromFsLib 59
	save calling environment in	jmp_buf argument (ANSI).....	setjmp() 1201
	announce clock tick to	kernel.....	tickAnnounce() 1379

	Keyword	Name	Page
	initialize kernel.	kernelInit()	726
	<b>netPoolInit()</b> , kernel heap version of	netPoolKheapInit()	927
/MB86930 automatic locking of	kernel instructions/data.	cacheMb930LockAuto()	459
VxWorks	kernel library.	kernelLib	129
return	kernel revision string.	kernelVersion()	727
add hook to bypass RIP and	kernel routing tables.	ripLeakHookAdd()	1096
get value of	kernel's tick counter.	tickGet()	1379
set value of	kernel's tick counter.	tickSet()	1380
add new RIP authentication	key.	ripAuthKeyAdd()	1088
existing RIP authentication	key. delete	ripAuthKeyDelete()	1088
find RIP authentication	key.	ripAuthKeyFind()	1089
find RIP authentication	key.	ripAuthKeyFindFirst()	1089
node that matches specified	key. find hash	hashTblFind()	648
create thread specific data	key (POSIX).	pthread_key_create()	1029
delete thread specific data	key (POSIX).	pthread_key_delete()	1029
compare	keys as 32 bit identifiers.	hashKeyCmp()	645
point to. compare	keys based on strings they	hashKeyStrCmp()	645
return very	large double.	infinity()	681
return very	large float.	infinityf()	682
find size of	largest available free block.	memPartFindMax()	815
memory system partition/ find	largest free block in shared	smMemFindMax()	1233
memory partition. find	largest free block in system	memFindMax()	811
equal to specified/ compute	largest integer less than or	floor()	597
equal to specified/ compute	largest integer less than or	floorf()	597
ifStackTable interface has no	layers above. test if	stackEntryIsTop()	1267
test if interface has no	layers beneath it.	stackEntryIsBottom()	1267
two strings as appropriate to	LC_COLLATE (ANSI). compare.	strcoll()	1272
components of object with type	lconv (ANSI). set	localeconv()	742
relinquish specified	lease.	dhcpcRelease()	534
renew established	lease.	dhcpcVerify()	538
routine to store and retrieve	lease data. add	dhcpcCacheHookAdd()	523
delete	lease data storage routine.	dhcpcCacheHookDelete()	524
Internet address. extract	lease information from	bootLeaseExtract()	433
display current	lease parameters.	dhcpcParamsShow()	534
network interface and setup	lease request. assign.	dhcpcInit()	527
assign permanent	lease storage hook for server.	dhcpcLeaseHookAdd()	541
retrieve current	lease timers.	dhcpcTimerGet()	537
display current	lease timers.	dhcpcTimersShow()	537
determine	length in bytes of zbuf.	zbufLength()	1499
(Unimplemented)/ calculate	length of multibyte character	mblen()	804
determine	length of string (ANSI).	strlen()	1277
determine	length of zbuf segment.	zbufSegLength()	1501
from given set/ return string	length up to first character	strcspn()	1273
not in given/ return string	length up to first character	strspn()	1280
test whether character is	letter (ANSI).	isalpha()	720
character is lower-case	letter (ANSI). test whether.	islower()	722
character is upper-case	letter (ANSI). test whether.	isupper()	724
equivalent/ convert upper-case	letter to lower-case.	tolower()	1392
equivalent/ convert lower-case	letter to upper-case	toupper()	1392
compare two strings	lexicographically (ANSI).	strcmp()	1272

	Keyword	Name	Page
change	line-delete character.....	tyDeleteLineSet()	1407
read line with	line-editing.....	ledRead()	734
	line-editing library.....	ledLib	131
discard	line-editor ID.....	ledClose()	733
create new	line-editor ID.....	ledOpen()	734
change	line-editor ID parameters.....	ledControl()	733
add physical address into	linked list.....	rcvEtherAddrAdd()	1058
library. doubly	linked list subroutine.....	lstLib	137
(C++). call all	linked static constructors.....	cplusCtorsLink()	504
(C++). call all	linked static destructors.....	cplusDtorsLink()	506
form frame with	link-layer address.....	muxAddressForm()	870
attach	link-level header to packet.....	muxLinkHeaderCreate()	881
physical address into linked	list. add.....	rcvEtherAddrAdd()	1058
add interface to RIP exclusion	list.....	ripIfExcludeListAdd()	1093
interface from RIP exclusion	list. delete.....	ripIfExcludeListDelete()	1094
show RIP interface exclusion	list.....	ripIfExcludeListShow()	1094
add new interfaces to internal	list.....	ripIfSearch()	1095
add directory to access	list.....	tftpdDirectoryAdd()	1369
delete directory from access	list.....	tftpdDirectoryRemove()	1369
add new trigger to trigger	list.....	trgAdd()	1393
delete trigger from trigger	list.....	trgDelete()	1395
add option to option request	list.....	dhcpcOptionSet()	531
address to multicast address	list. add multicast.....	etherMultiAdd()	573
find I/O device in device	list.....	iosDevFind()	709
add node to end of	list.....	lstAdd()	761
report number of nodes in	list.....	lstCount()	762
delete specified node from	list.....	lstDelete()	763
extract sublist from	list.....	lstExtract()	763
find node in	list.....	lstFind()	764
find first node in	list.....	lstFirst()	764
free up	list.....	lstFree()	765
and return first node from	list. delete.....	lstGet()	765
find last node in	list.....	lstLast()	767
find next node in	list.....	lstNext()	768
find Nth node in	list.....	lstNth()	769
find previous node in	list.....	lstPrevious()	769
insert node in	list after specified node.....	lstInsert()	766
	list all system-known devices.....	devs()	519
any of subdirectories.	list contents of directory and.....	lsr()	761
(multi-purpose).	list contents of directory.....	dirList()	542
initialize	list descriptor.....	lstInit()	766
FTP.	list directory contents via.....	ftpLs()	627
from specified node. find	list node nStep steps away.....	lstNStep()	768
to SCSI controller.	list physical devices attached.....	scsiShow()	1160
adds device to	list polled by tMuxPollTask.....	muxPollDevAdd()	887
removes device from	list polled by tMuxPollTask.....	muxPollDevDel()	887
reports whether device is on	list polled by tMuxPollTask.....	muxPollDevStat()	888
doubly linked	list subroutine library.....	lstLib	137
	list symbols.....	lkup()	737
near specified value.	list symbols whose values are.....	lkAddr()	736

	Keyword	Name	Page
MIB-II entry from UDP list of	listeners. get UDP .....	<b>m2UdpTblEntryGet()</b>	801
concatenate two	lists. ....	<b>lstConcat()</b>	762
/with variable argument	list to buffer (ANSI). ....	<b>vsprintf()</b>	1446
with variable argument	list to fd. /string formatted .....	<b>vfdprintf()</b>	1430
/with variable argument	list to standard output/ .....	<b>vprintf()</b>	1446
	load driver into MUX. ....	<b>muxDevLoad()</b>	876
memory.	load object module into.....	<b>ld()</b>	730
memory.	load object module into.....	<b>loadModule()</b>	739
memory.	load object module into.....	<b>loadModuleAt()</b>	739
whether device is already	loaded into MUX. tests.....	<b>muxDevExists()</b>	876
get list of	loaded modules. ....	<b>moduleIdListGet()</b>	833
show current status for all	loaded modules. ....	<b>moduleShow()</b>	836
object module	loader. ....	<b>loadLib</b>	133
device. issue	LOAD/UNLOAD command to SCSI .....	<b>scsiLoadUnit()</b>	1145
convert bus address to	local address. ....	<b>sysBusToLocalAdrs()</b>	1301
from Internet address. get	local address ((host number)).....	<b>inet_lnaof()</b>	676
convert	local address to bus address.....	<b>sysLocalToBusAdrs()</b>	1306
address (VxMP). convert	local address to global.....	<b>smObjLocalToGlobal()</b>	1245
convert global address to	local address (VxMP). ....	<b>smObjGlobalToLocal()</b>	1243
find object by name in	local database (VxFusion).....	<b>distNameFind()</b>	553
initialize	local debugging package. ....	<b>dbgInit()</b>	514
Register/ get content of	Local Descriptor Table.....	<b>vxLdtrGet()</b>	1453
set appropriate	locale (ANSI).....	<b>setlocale()</b>	1203
ANSI	locale documentation. ....	<b>ansiLocale</b>	15
send advertisement to one	location. ....	<b>sendAdvert()</b>	1196
containing specified byte	location. find zbuf segment .....	<b>zbufSegFind()</b>	1500
test and set	location across bus. ....	<b>sysBusTas()</b>	1301
segment. determine	location of data in zbuf .....	<b>zbufSegData()</b>	1499
copy memory from one	location to another (ANSI).....	<b>memcpy()</b>	808
copy memory from one	location to another (ANSI).....	<b>memmove()</b>	812
advertisement to all active	locations. send .....	<b>sendAdvertAll()</b>	1197
	lock access to shell. ....	<b>shellLock()</b>	1213
cache.	lock all or part of specified .....	<b>cacheLock()</b>	458
into memory (POSIX).	lock all pages used by process .....	<b>mlockall()</b>	823
(POSIX).	lock mutex if it is available .....	<b>pthread_mutex_trylock()</b>	1033
	lock mutex (POSIX). ....	<b>pthread_mutex_lock()</b>	1032
	lock out interrupts. ....	<b>intLock()</b>	691
memory (POSIX).	lock specified pages into .....	<b>mlock()</b>	822
blocking if not available/	lock (take) semaphore, .....	<b>sem_wait()</b>	1195
returning error if/	lock (take) semaphore, .....	<b>sem_trywait()</b>	1194
enable MB86930 automatic	locking of kernel/ .....	<b>cacheMb930LockAuto()</b>	459
SH,/ get current interrupt	lock-out level (68K, x86, ARM, .....	<b>intLockLevelGet()</b>	693
SH,/ set current interrupt	lock-out level (68K, x86, ARM, .....	<b>intLockLevelSet()</b>	693
cancel interrupt	locks. ....	<b>intUnlock()</b>	696
	log formatted error message. ....	<b>logMsg()</b>	757
transfer	log header to host (WindView). ....	<b>wvLogHeaderUpload()</b>	1474
	log in to remote FTP server.....	<b>ftpLogin()</b>	627
	log in to remote host. ....	<b>rlogin()</b>	1105
	log out of VxWorks system.....	<b>logout()</b>	758
(WindView).	log user-defined event.....	<b>wvEvent()</b>	1469

	Keyword	Name	Page
set class of events to	log (WindView).....	<b>wvEvtClassSet()</b>	1471
initialize event	log (WindView).....	<b>wvEvtLogInit()</b>	1472
compute base-2	logarithm.....	<b>log2()</b>	747
compute base-2	logarithm.....	<b>log2f()</b>	747
compute natural	logarithm (ANSI).....	<b>log()</b>	746
compute base-10	logarithm (ANSI).....	<b>log10()</b>	748
compute base-10	logarithm (ANSI).....	<b>log10f()</b>	748
compute natural	logarithm (ANSI).....	<b>logf()</b>	749
of events from those being	logged (WindView). /class .....	<b>wvEvtClassClear()</b>	1470
classes of events from those	logged (WindView). clear all .....	<b>wvEvtClassClearAll()</b>	1470
current set of classes being	logged (WindView). get.....	<b>wvEvtClassGet()</b>	1471
(WindView). event	logging control library .....	<b>wvLib</b>	351
(WindView). start	logging events to buffer.....	<b>wvEvtLogStart()</b>	1472
(WindView). stop	logging events to buffer.....	<b>wvEvtLogStop()</b>	1473
add	logging fd. ....	<b>logFdAdd()</b>	749
delete	logging fd. ....	<b>logFdDelete()</b>	750
set primary	logging fd. ....	<b>logFdSet()</b>	750
message	logging library.....	<b>logLib</b>	136
initialize message	logging library.....	<b>logInit()</b>	753
take spin-lock/ control	logging of failed attempts to.....	<b>smObjTimeoutLogEnable()</b>	1248
get address of top of	logical memory.....	<b>sysMemTop()</b>	1307
initialize fields in SCSI	logical partition.....	<b>scsiBlkDevInit()</b>	1138
block device. define	logical partition on SCSI.....	<b>scsiBlkDevCreate()</b>	1137
VxWorks remote	login daemon. ....	<b>rlogind()</b>	1105
initialize remote	login facility.....	<b>rlogInit()</b>	1106
remote	login library. ....	<b>rlogLib</b>	236
entry. display	login prompt and validate user.....	<b>loginPrompt()</b>	753
change	login string. ....	<b>loginStringSet()</b>	754
initialize	login table. ....	<b>loginInit()</b>	752
add user to	login table. ....	<b>loginUserAdd()</b>	754
delete user entry from	login table. ....	<b>loginUserDelete()</b>	755
display user	login table. ....	<b>loginUserShow()</b>	756
user name and password in	login table. verify.....	<b>loginUserVerify()</b>	756
library. user	login/password subroutine.....	<b>loginLib</b>	134
print VxWorks	logo.....	<b>printLogo()</b>	1001
convert string to	long (ANSI). ....	<b>atol()</b>	422
compute absolute value of	long (ANSI). ....	<b>labs()</b>	729
notation Internet address to	long integer. convert dot .....	<b>inet_addr()</b>	675
convert string to	long integer (ANSI).....	<b>strtol()</b>	1284
convert string to unsigned	long integer (ANSI).....	<b>strtoul()</b>	1285
all subdirectories/ do	long listing of directory and .....	<b>llr()</b>	738
contents. generate	long listing of directory .....	<b>ll()</b>	737
copy one buffer to another one	long word at a time. ....	<b>bcopyLongs()</b>	426
flush TLBs (Translation	Lookaside Buffers). ....	<b>pentiumTlbFlush()</b>	973
components.	low level I/O access to flash.....	<b>tffsRawio()</b>	1366
(Timestamp Counter). get	lower half of 64Bit TSC.....	<b>pentiumTscGet32()</b>	974
convert upper-case letter to	lower-case equivalent (ANSI).....	<b>tolower()</b>	1392
test whether character is	lower-case letter (ANSI). ....	<b>islower()</b>	722
upper-case equivalent/ convert	lower-case letter to .....	<b>toupper()</b>	1392
initializes	<b>lstLib</b> module. ....	<b>lstLibInit()</b>	767

	Keyword	Name	Page
of system register	mach (also macl, pr) (SH). /contents .....	<b>mach()</b>	802
connect routine to	mailbox interrupt. ....	<b>sysMailboxConnect()</b>	1306
enable	mailbox interrupt. ....	<b>sysMailboxEnable()</b>	1307
error message (ANSI).	map error number in errno to .....	<b>perorr()</b>	976
string (ANSI).	map error number to error .....	<b>strerror()</b>	1274
string (POSIX).	map error number to error .....	<b>strerror_r()</b>	1274
interface structure pointer.	map interface name to .....	<b>ifunit()</b>	671
space in shared global/	map physical pages to virtual .....	<b>vmGlobalMap()</b>	1438
virtual space (VxVMI).	map physical space into .....	<b>vmMap()</b>	1440
initialize global	mapping. ....	<b>vmBaseGlobalMapInit()</b>	1431
processors. MMU	mapping library for ARM Ltd. ....	<b>mmuMapLib</b>	162
initialize global	mapping (VxVMI).....	<b>vmGlobalMapInit()</b>	1439
device. write file	marks to SCSI sequential .....	<b>scsiWrtFileMarks()</b>	1167
set signal	mask. ....	<b>sigsetmask()</b>	1223
address. extract net	mask field from Internet .....	<b>bootNetmaskExtract()</b>	434
get subnet	mask for network interface.....	<b>ifMaskGet()</b>	668
calling thread's signal	mask (POSIX). /and/or examine .....	<b>pthread_sigmask()</b>	1042
examine and/or change signal	mask (POSIX).....	<b>sigprocmask()</b>	1221
find hash node that	matches specified key. ....	<b>hashTblFind()</b>	648
interface library for multiple	matching entries. route .....	<b>routeEntryLib</b>	238
destination. find	matching route for .....	<b>routeEntryLookup()</b>	1118
ANSI	math documentation. ....	<b>ansiMath</b>	15
library to high-level	math functions. C interface .....	<b>mathALib</b>	153
hardware floating-point	math library.....	<b>mathHardLib</b>	155
hardware floating-point	math support. initialize.....	<b>mathHardInit()</b>	803
software floating-point	math support. initialize.....	<b>mathSoftInit()</b>	803
get	maximum priority (POSIX). ....	<b>sched_get_priority_max()</b>	1131
kernel/ enable	MB86930 automatic locking of.....	<b>cacheMb930LockAuto()</b>	459
clear line from	MB86930 cache. ....	<b>cacheMb930ClearLine()</b>	458
initialize Fujitsu	MB86930 cache library.....	<b>cacheMb930LibInit()</b>	459
and join it to specified	mBlk. get cBlk-cluster.....	<b>netMblkClGet()</b>	918
duplicate	mBlk. ....	<b>netMblkDup()</b>	920
free	mBlk back to memory pool. ....	<b>netMblkFree()</b>	921
duplicate	mBlk chain. ....	<b>netMblkChainDup()</b>	916
get	mBlk from memory pool. ....	<b>netMblkGet()</b>	921
copy data from	mBlk to buffer. ....	<b>netMblkToBufCopy()</b>	922
construct. join	mBlk to cBlk-cluster .....	<b>netMblkClJoin()</b>	919
get	mBlk-cBlk-cluster.....	<b>netTupleGet()</b>	930
free	mBlk-cBlk-cluster construct. ....	<b>netMblkClFree()</b>	917
free chain of	mBlk-cBlk-cluster constructs.....	<b>netMblkClChainFree()</b>	917
report	mbuf statistics.....	<b>mbufShow()</b>	805
disable superscalar dispatch	(MC68060). ....	<b>vxSSDisable()</b>	1458
enable superscalar dispatch	(MC68060). ....	<b>vxSSEnable()</b>	1458
disable store buffer	(MC68060 only). ....	<b>cacheStoreBufDisable()</b>	470
enable store buffer	(MC68060 only). ....	<b>cacheStoreBufEnable()</b>	470
Architecture). enable/disable	MCA (Machine Check.....	<b>pentiumMcaEnable()</b>	953
Architecture) registers. show	MCA (Machine Check.....	<b>pentiumMcaShow()</b>	953
incoming RIP-2 message using	MD5. authenticate.....	<b>ripAuthKeyInMD5()</b>	1090
outgoing RIP-2 message using	MD5. authenticate.....	<b>ripAuthKeyOut2MD5()</b>	1091
RIP-2 message. start	MD5 authentication of outgoing .....	<b>ripAuthKeyOut1MD5()</b>	1090

	Keyword	Name	Page
	detect change in media.....	<b>scsiSeqStatusCheck()</b>	1160
formatting library. MS-DOS	media-compatible file system.....	<b>dosFsFmtLib</b>	86
library. RT-11	media-compatible file system.....	<b>rt11FsLib</b>	239
library. MS-DOS	media-compatible file system.....	<b>dosFsLib</b>	86
space in shared global virtual	mem (VxVMI). /pages to virtual.....	<b>vmGlobalMap()</b>	1438
display all or one group with	members (VxFusion).....	<b>msgQDistGrpShow()</b>	854
get address of top of logical	memory.....	<b>sysMemTop()</b>	1307
get address of top of	memory.....	<b>sysPhysMemTop()</b>	1310
processor write buffers to	memory. flush.....	<b>cachePipeFlush()</b>	460
transfer blocks to or from	memory.....	<b>cbioBlkRW()</b>	477
transfer bytes to or from	memory.....	<b>cbioBytesRW()</b>	477
free block of	memory.....	<b>cfree()</b>	490
display	memory.....	<b>d()</b>	512
load object module into	memory.....	<b>ld()</b>	730
load object module into	memory.....	<b>loadModule()</b>	739
load object module into	memory.....	<b>loadModuleAt()</b>	739
modify	memory.....	<b>m()</b>	770
allocate aligned	memory.....	<b>memalign()</b>	806
reallocate block of	memory (ANSI).....	<b>realloc()</b>	1065
free block of	memory (ANSI).....	<b>free()</b>	612
compare two blocks of	memory (ANSI).....	<b>memcmp()</b>	807
set block of	memory (ANSI).....	<b>memset()</b>	819
create and initialize shared	memory binary semaphore/.....	<b>semBSmCreate()</b>	1175
create and initialize shared	memory counting semaphore/.....	<b>semCSmCreate()</b>	1178
default run-time support for	memory deallocation (C++).....	<b>operator delete()</b>	946
create	memory device.....	<b>memDevCreate()</b>	808
delete	memory device.....	<b>memDevDelete()</b>	810
files. create	memory device for multiple.....	<b>memDevCreateDir()</b>	810
install	memory driver.....	<b>memDrv()</b>	811
memory system/ allocate	memory for array from shared.....	<b>smMemCalloc()</b>	1232
search block of	memory for character (ANSI).....	<b>memchr()</b>	807
another (ANSI). copy	memory from one location to.....	<b>memcpy()</b>	808
another (ANSI). copy	memory from one location to.....	<b>memmove()</b>	812
allocate aligned	memory from partition.....	<b>memPartAlignedAlloc()</b>	813
allocate block of	memory from partition.....	<b>memPartAlloc()</b>	814
system/ allocate block of	memory from shared memory.....	<b>smMemMalloc()</b>	1234
system/ reallocate block of	memory from shared memory.....	<b>smMemRealloc()</b>	1235
partition/ allocate block of	memory from system memory.....	<b>malloc()</b>	802
free block of	memory in partition.....	<b>memPartFree()</b>	815
reallocate block of	memory in specified partition.....	<b>memPartRealloc()</b>	817
(POSIX).	memory management library.....	<b>mmanPxLib</b>	162
(VxMP). shared	memory management library.....	<b>smMemLib</b>	281
routines (VxMP). shared	memory management show.....	<b>smMemShow</b>	284
(VxMP). shared	memory message queue library.....	<b>msgQSmLib</b>	185
create and initialize shared	memory message queue (VxMP).....	<b>msgQSmCreate()</b>	868
add name to shared	memory name database (VxMP).....	<b>smNameAdd()</b>	1237
show information about shared	memory network.....	<b>smNetShow()</b>	1241
VxWorks interface to shared	memory network (backplane)/.....	<b>smNetLib</b>	287
routines. shared	memory network driver show.....	<b>smNetShow</b>	288
look up shared	memory object by name (VxMP).....	<b>smNameFind()</b>	1238

	Keyword	Name	Page
look up shared	memory object by value (VxMP).....	<b>smNameFindByValue()</b>	1239
(VxMP). initialize shared	memory objects descriptor .....	<b>smObjInit()</b>	1244
attach calling CPU to shared	memory objects facility/ .....	<b>smObjAttach()</b>	1242
(VxMP). install shared	memory objects facility .....	<b>smObjLibInit()</b>	1245
(VxMP). initialize shared	memory objects facility .....	<b>smObjSetup()</b>	1246
shared	memory objects library (VxMP).....	<b>smObjLib</b>	288
library (VxMP). shared	memory objects name database .....	<b>smNameLib</b>	284
show routines (VxMP). shared	memory objects name database .....	<b>smNameShow</b>	286
remove object from shared	memory objects name database/ .....	<b>smNameRemove()</b>	1240
show contents of shared	memory objects name database/ .....	<b>smNameShow()</b>	1240
(VxMP). shared	memory objects show routines .....	<b>smObjShow</b>	291
/current status of shared	memory objects (VxMP). .....	<b>smObjShow()</b>	1247
allocate	memory on page boundary. ....	<b>valloc()</b>	1429
add memory to system	memory partition. ....	<b>memAddToPool()</b>	806
largest free block in system	memory partition. find.....	<b>memFindMax()</b>	811
set debug options for system	memory partition. ....	<b>memOptionsSet()</b>	812
add memory to	memory partition. ....	<b>memPartAddToPool()</b>	813
create	memory partition. ....	<b>memPartCreate()</b>	814
set debug options for	memory partition. ....	<b>memPartOptionsSet()</b>	816
/block of memory from system	memory partition (ANSI). ....	<b>malloc()</b>	802
statistics. show system	memory partition blocks and .....	<b>memShow()</b>	820
full-featured	memory partition manager.....	<b>memLib</b>	158
core	memory partition manager.....	<b>memPartLib</b>	160
facility. initialize	memory partition show .....	<b>memShowInit()</b>	821
create shared	memory partition (VxMP). ....	<b>memPartSmCreate()</b>	818
construct back to	memory pool. /clBlk-cluster .....	<b>netClBlkFree()</b>	907
free cluster back to	memory pool. ....	<b>netClFree()</b>	909
free mBlk back to	memory pool. ....	<b>netMblkFree()</b>	921
get mBlk from	memory pool. ....	<b>netMblkGet()</b>	921
delete	memory pool. ....	<b>netPoolDelete()</b>	923
initialize netBufLib-managed	memory pool. ....	<b>netPoolInit()</b>	923
lock specified pages into	memory (POSIX). ....	<b>mlock()</b>	822
all pages used by process into	memory (POSIX). lock.....	<b>mlockall()</b>	823
(VxMP). shared	memory semaphore library .....	<b>semSmLib</b>	274
	memory show routines. ....	<b>memShow</b>	161
of memory (VxMP). free shared	memory system partition block .....	<b>smMemFree()</b>	1233
and statistics/ show shared	memory system partition blocks .....	<b>smMemShow()</b>	1236
(VxMP). add memory to shared	memory system partition.....	<b>smMemAddToPool()</b>	1231
/memory for array from shared	memory system partition/ .....	<b>smMemCalloc()</b>	1232
largest free block in shared	memory system partition/ find.....	<b>smMemFindMax()</b>	1233
/block of memory from shared	memory system partition/ .....	<b>smMemMalloc()</b>	1234
set debug options for shared	memory system partition/ .....	<b>smMemOptionsSet()</b>	1234
/block of memory from shared	memory system partition/ .....	<b>smMemRealloc()</b>	1235
add	memory to memory partition. ....	<b>memPartAddToPool()</b>	813
partition (VxMP). add	memory to shared memory system.....	<b>smMemAddToPool()</b>	1231
partition. add	memory to system memory.....	<b>memAddToPool()</b>	806
disable MTRR	(Memory Type Range Register). ....	<b>pentiumMtrrDisable()</b>	956
enable MTRR	(Memory Type Range Register). ....	<b>pentiumMtrrEnable()</b>	956
system partition block of	memory (VxMP). /shared memory .....	<b>smMemFree()</b>	1233
display task monitoring help	menu. ....	<b>spyHelp()</b>	1258

	Keyword	Name	Page
display debugging help	menu.....	dbgHelp()	513
display NFS help	menu.....	nfsHelp()	940
address pointers from route	message. extract socket.....	ripAdrrsXtract()	1083
of outgoing RIP-2	message. /MD5 authentication.....	ripAuthKeyOut1MD5()	1090
build identification	message.....	scsiIdentMsgBuild()	1143
parse identification	message.....	scsiIdentMsgParse()	1143
log formatted error	message.....	logMsg()	757
send BOOTP request	message and retrieve reply.....	bootpMsgGet()	436
socket. create zbuf from user	message and send it to UDP.....	zbufSockBufSendto()	1503
error number in errno to error	message (ANSI). map.....	perorr()	976
message queue/ receive	message from distributed.....	msgQDistReceive()	856
receive	message from message queue.....	msgQReceive()	864
(POSIX). receive	message from message queue.....	mq_receive()	842
receive	message from socket.....	recvfrom()	1068
receive	message from socket.....	recvmsg()	1069
socket. receive	message in zbuf from UDP.....	zbufSockRecvfrom()	1506
(POSIX). notify task that	message is available on queue.....	mq_notify()	840
post-processing when outgoing	message is rejected. perform.....	scsiMsgOutReject()	1151
post-processing after SCSI	message is sent. perform.....	scsiMsgOutComplete()	1150
	message logging library.....	logLib	136
initialize	message logging library.....	logInit()	753
create and initialize	message queue.....	msgQCreate()	850
delete	message queue.....	msgQDelete()	851
event notification process for	message queue. start.....	msgQEvStart()	859
event notification process for	message queue. stop.....	msgQEvStop()	860
get information about	message queue.....	msgQInfoGet()	861
number of messages queued to	message queue. get.....	msgQNumMsgs()	863
receive message from	message queue.....	msgQReceive()	864
send message to	message queue.....	msgQSend()	865
show information about	message queue.....	msgQShow()	866
(POSIX). get	message queue attributes.....	mq_getattr()	839
(POSIX). set	message queue attributes.....	mq_setattr()	844
delete distributed	message queue from group/.....	msgQDistGrpDelete()	854
(VxFusion). distributed	message queue group library.....	msgQDistGrpLib	180
routines/ distributed	message queue group show.....	msgQDistGrpShow	181
	message queue library.....	msgQLib	183
initialize POSIX	message queue library.....	mqPxLibInit()	837
	message queue library (POSIX).....	mqPxLib	179
distributed objects	message queue library /.....	msgQDistLib	181
shared memory	message queue library (VxMP).....	msgQSmLib	185
close	message queue (POSIX).....	mq_close()	838
open	message queue (POSIX).....	mq_open()	841
receive message from	message queue (POSIX).....	mq_receive()	842
send message to	message queue (POSIX).....	mq_send()	843
remove	message queue (POSIX).....	mq_unlink()	845
POSIX	message queue show.....	mqPxShow	180
initialize POSIX	message queue show facility.....	mqPxShowInit()	838
initialize	message queue show facility.....	msgQShowInit()	867
initialize distributed	message queue show package/.....	msgQDistShowInit()	859
	message queue show routines.....	msgQShow	185

	Keyword	Name	Page
(VxFusion). distributed	message queue show routines .....	<b>msgQDistShow</b>	182
(VxFusion). add distributed	message queue to group .....	<b>msgQDistGrpAdd()</b>	853
create distributed	message queue (VxFusion).....	<b>msgQDistCreate()</b>	851
/of messages in distributed	message queue (VxFusion).....	<b>msgQDistNumMsgs()</b>	855
/message from distributed	message queue (VxFusion).....	<b>msgQDistReceive()</b>	856
send message to distributed	message queue (VxFusion).....	<b>msgQDistSend()</b>	857
and initialize shared memory	message queue (VxMP). create .....	<b>msgQSmCreate()</b>	868
VxWorks events support for	message queues.....	<b>msgQEvLib</b>	183
handle complete SCSI	message received from target.....	<b>scsiMsgInComplete()</b>	1150
interface library.	message routines for routing.....	<b>routeMessageLib</b>	238
queue (VxFusion). send	message to distributed message .....	<b>msgQDistSend()</b>	857
send	message to message queue.....	<b>msgQSend()</b>	865
(POSIX). send	message to message queue.....	<b>mq_send()</b>	843
send TFTP	message to remote system.....	<b>tftpSend()</b>	1375
send	message to socket.....	<b>sendmsg()</b>	1197
send	message to socket.....	<b>sendto()</b>	1198
send zbuf	message to UDP socket.....	<b>zbufSockSendto()</b>	1508
authenticate incoming RIP-2	message using MD5.....	<b>ripAuthKeyInMD5()</b>	1090
authenticate outgoing RIP-2	message using MD5.....	<b>ripAuthKeyOut2MD5()</b>	1091
	message-logging support task.....	<b>logTask()</b>	758
add option to client	messages.....	<b>dhcpcOptionAdd()</b>	529
message queue/ get number of	messages in distributed .....	<b>msgQDistNumMsgs()</b>	855
queue. get number of	messages queued to message.....	<b>msgQNumMsgs()</b>	863
get	metric for network interface.....	<b>ifMetricGet()</b>	669
helper file for igmp	Mib.....	<b>m2Igmplib</b>	141
delete RIP	MIB support.....	<b>m2RipDelete()</b>	792
initialize RIP	MIB support.....	<b>m2RipInit()</b>	795
initialize SNMP	MIB-2 library.....	<b>m2Init()</b>	786
get	MIB-2 routing table entry.....	<b>m2IpRouteTblEntryGet()</b>	791
get IP	MIB-II address entry.....	<b>m2IpAddrTblEntryGet()</b>	787
agents.	MIB-II API library for SNMP .....	<b>m2Lib</b>	144
add, modify, or delete	MIB-II ARP entry.....	<b>m2IpAtransTblEntrySet()</b>	788
get	MIB-II ARP table entry.....	<b>m2IpAtransTblEntryGet()</b>	787
listeners. get UDP	MIB-II entry from UDP list of .....	<b>m2UdpTblEntryGet()</b>	801
initialize	MIB-II ICMP-group access.....	<b>m2IcmpInit()</b>	772
Agents.	MIB-II ICMP-group API for SNMP .....	<b>m2IcmpLib</b>	139
variables. get	MIB-II ICMP-group global .....	<b>m2IcmpGroupInfoGet()</b>	771
or DOWN. set state of	MIB-II interface entry to UP .....	<b>m2IfTblEntrySet()</b>	784
SNMP agents.	MIB-II interface-group API for .....	<b>m2IfLib</b>	139
routines. initialize	MIB-II interface-group .....	<b>m2IfInit()</b>	778
variables. get	MIB-II interface-group scalar .....	<b>m2IfGroupInfoGet()</b>	778
entry. get	MIB-II interface-group table.....	<b>m2IfStackEntryGet()</b>	781
entry. get	MIB-II interface-group table.....	<b>m2IfTblEntryGet()</b>	783
initialize	MIB-II IP-group access.....	<b>m2IpInit()</b>	790
agents.	MIB-II IP-group API for SNMP .....	<b>m2IpLib</b>	142
variables. get	MIB-II IP-group scalar.....	<b>m2IpGroupInfoGet()</b>	789
new values. set	MIB-II IP-group variables to .....	<b>m2IpGroupInfoSet()</b>	790
delete all	MIB-II library groups.....	<b>m2Delete()</b>	770
counters. get	MIB-II RIP-group global .....	<b>m2RipGlobalCountersGet()</b>	793
entry. get	MIB-II RIP-group interface.....	<b>m2RipIfConfEntryGet()</b>	793

	Keyword	Name	Page
	entry. set	MIB-II RIP-group interface ..... <b>m2RipIfConfEntrySet()</b>	794
	entry. get	MIB-II RIP-group interface ..... <b>m2RipIfStatEntryGet()</b>	794
	set	MIB-II routing table entry. .... <b>m2IpRouteTblEntrySet()</b>	791
	resources used to access	MIB-II system group. delete ..... <b>m2SysDelete()</b>	795
	SNMP agents.	MIB-II system-group API for..... <b>m2SysLib</b>	148
	initialize	MIB-II system-group routines. .... <b>m2SysInit()</b>	797
	entry. get	MIB-II TCP connection table..... <b>m2TcpConnEntryGet()</b>	797
	initialize	MIB-II TCP-group access. .... <b>m2TcpInit()</b>	799
	agents.	MIB-II TCP-group API for SNMP..... <b>m2TcpLib</b>	150
	variables. get	MIB-II TCP-group scalar ..... <b>m2TcpGroupInfoGet()</b>	799
	initialize	MIB-II UDP-group access..... <b>m2UdpInit()</b>	801
	agents.	MIB-II UDP-group API for SNMP ..... <b>m2UdpLib</b>	152
	variables. get	MIB-II UDP-group scalar ..... <b>m2UdpGroupInfoGet()</b>	800
	get system-group	MIB-II variables. .... <b>m2SysGroupInfoGet()</b>	796
	values. set system-group	MIB-II variables to new ..... <b>m2SysGroupInfoSet()</b>	796
	get	minimum priority (POSIX). .... <b>sched_get_priority_min()</b>	1132
	default task status register	(MIPS). initialize ..... <b>taskSRInit()</b>	1346
	library.	MIPS 4kc cache management ..... <b>cache4kcLib</b>	37
	handler to breakpoint type	(MIPS). bind breakpoint ..... <b>dbgBpTypeBind()</b>	513
	contents of cause register	(MIPS). read..... <b>intCRGet()</b>	686
	contents of cause register	(MIPS). write ..... <b>intCRSet()</b>	687
	contents of status register	(MIPS). read..... <b>intSRGet()</b>	694
	contents of status register	(MIPS). update..... <b>intSRSet()</b>	694
	/(trap) base address (68K, x86,	MIPS, ARM, SimSolaris, SimNT). .... <b>intVecBaseGet()</b>	696
	/(trap) base address (68K, x86,	MIPS, ARM, SimSolaris, SimNT). .... <b>intVecBaseSet()</b>	697
	corresponding interrupt bits	(MIPS, PowerPC, ARM). disable..... <b>intDisable()</b>	687
	corresponding interrupt bits	(MIPS, PowerPC, ARM). enable..... <b>intEnable()</b>	688
	library.	MIPS R10000 cache management..... <b>cacheR10kLib</b>	49
	library.	MIPS R3000 cache management..... <b>cacheR3kLib</b>	47
	library.	MIPS R33000 cache management..... <b>cacheR33kLib</b>	50
	library.	MIPS R333x0 cache management ..... <b>cacheR333x0Lib</b>	50
	library.	MIPS R4000 cache management..... <b>cacheR4kLib</b>	47
	library.	MIPS R5000 cache management..... <b>cacheR5kLib</b>	48
	library.	MIPS R7000 cache management..... <b>cacheR7kLib</b>	48
	library.	MIPS RC32364 cache management..... <b>cacheR32kLib</b>	49
	/interrupt vector (68K, x86,	MIPS, SH, SimSolaris, SimNT). .... <b>intVecGet()</b>	698
	/CPU vector (trap) (68K, x86,	MIPS, SH, SimSolaris, SimNT). .... <b>intVecSet()</b>	699
	for C routine (68K, x86,	MIPS, SimSolaris). /handler..... <b>intHandlerCreate()</b>	688
	set task status register (68K,	MIPS, x86)..... <b>taskSRSet()</b>	1347
	routines.	miscellaneous support..... <b>vxLib</b>	346
	PentiumPro/2/3/4 32 bit mode.	MMU library for ..... <b>mmuPro32Lib</b>	163
	Ltd. processors.	MMU mapping library for ARM ..... <b>mmuMapLib</b>	162
	Hitachi SH7700	MMU support library..... <b>mmuSh7700Lib</b>	168
	Hitachi SH7750	MMU support library..... <b>mmuSh7750Lib</b>	172
	return	model name of CPU board. .... <b>sysModel()</b>	1308
	get contents of specified MSR	(Model Specific Register)..... <b>pentiumMsrGet()</b>	954
	initialize all MSRs	(Model Specific Register)..... <b>pentiumMsrInit()</b>	954
	show all MSR	(Model Specific Register)..... <b>pentiumMsrShow()</b>	955
	set value to specified MSR	(Model Specific Registers)..... <b>pentiumMsrSet()</b>	955
	device. issue	MODE_SELECT command to SCSI..... <b>scsiModeSelect()</b>	1149

	Keyword	Name	Page
tape device. issue	MODE_SELECT command to SCSI .....	<b>scsiTapeModeSelect()</b>	1163
device. issue	MODE_SENSE command to SCSI.....	<b>scsiModeSense()</b>	1149
tape device. issue	MODE_SENSE command to SCSI.....	<b>scsiTapeModeSense()</b>	1163
transfer control to ROM	monitor. ....	<b>sysToMonitor()</b>	1317
show PMCs (Performance	Monitoring Counters). ....	<b>pentiumPmcShow()</b>	970
display task	monitoring help menu. ....	<b>spyHelp()</b>	1258
exported by specified host.	mount all file systems.....	<b>nfsMountAll()</b>	942
initialize	mount daemon. ....	<b>mountdInit()</b>	836
hard disk or CDROM.	mount DOS file system from ATA.....	<b>usrAtaConfig()</b>	1419
floppy disk.	mount DOS file system from.....	<b>usrFdConfig()</b>	1420
hard disk.	mount DOS file system from IDE.....	<b>usrIdeConfig()</b>	1424
	mount NFS file system. ....	<b>nfsMount()</b>	941
	mount protocol library. ....	<b>mountLib</b>	178
display	mounted NFS devices. ....	<b>nfsDevShow()</b>	935
format	MS-DOS compatible volume.....	<b>dosFsVolFormat()</b>	562
files. modify	MS-DOS file attributes of many.....	<b>xattrib()</b>	1489
or directory. modify	MS-DOS file attributes on file .....	<b>attrib()</b>	423
consistency checking on	MS-DOS file system. perform .....	<b>chkdsk()</b>	492
system formatting library.	MS-DOS media-compatible file .....	<b>dosFsFmtLib</b>	86
system library.	MS-DOS media-compatible file .....	<b>dosFsLib</b>	86
get contents of specified	MSR (Model Specific Register).....	<b>pentiumMsrGet()</b>	954
show all	MSR (Model Specific Register).....	<b>pentiumMsrShow()</b>	955
set value to specified	MSR (Model Specific/ .....	<b>pentiumMsrSet()</b>	955
Register). initialize all	MSRs (Model Specific.....	<b>pentiumMsrInit()</b>	954
Register). disable	MTRR (Memory Type Range .....	<b>pentiumMtrrDisable()</b>	956
Register). enable	MTRR (Memory Type Range .....	<b>pentiumMtrrEnable()</b>	956
get MTRRs to specified	MTRR table. ....	<b>pentiumMtrrGet()</b>	957
set MTRRs from specified	MTRR table with WRMSR/ .....	<b>pentiumMtrrSet()</b>	957
table with WRMSR/ set	MTRRs from specified MTRR .....	<b>pentiumMtrrSet()</b>	957
get	MTRRs to specified MTRR table.....	<b>pentiumMtrrGet()</b>	957
character/ convert	multibyte character to wide .....	<b>mbtowc()</b>	805
convert wide character to	multibyte character/ .....	<b>wctomb()</b>	1461
calculate length of	multibyte character/ .....	<b>mblen()</b>	804
char's/ convert series of	multibyte char's to wide.....	<b>mbstowcs()</b>	804
series of wide char's to	multibyte char's/ convert.....	<b>wcstombs()</b>	1461
device's multicast/ delete	multicast address from.....	<b>muxMCastAddrDel()</b>	883
add multicast address to	multicast address list. ....	<b>etherMultiAdd()</b>	573
delete Ethernet	multicast address record. ....	<b>etherMultiDel()</b>	574
MUX/Driver. get	multicast address table from .....	<b>muxMCastAddrGet()</b>	884
multicast table. add	multicast address to device's .....	<b>muxMCastAddrAdd()</b>	882
address list. add	multicast address to multicast .....	<b>etherMultiAdd()</b>	573
library to handle Ethernet	multicast addresses.....	<b>etherMultiLib</b>	104
driver. retrieve table of	multicast addresses from .....	<b>etherMultiGet()</b>	574
multicast address to device's	multicast table. add .....	<b>muxMCastAddrAdd()</b>	882
address from device's	multicast table. /multicast.....	<b>muxMCastAddrDel()</b>	883
function.	multiplicative hashing.....	<b>hashFuncMultiply()</b>	644
power of 2 (ANSI).	multiply number by integral.....	<b>ldexp()</b>	731
list contents of directory	(multi-purpose). ....	<b>dirList()</b>	542
/value of prioceiling attr in	mutex attr object (POSIX) ...	<b>pthread_mutexattr_getprioceiling()</b>	1035
set protocol attribute in	mutex attribute object/ .....	<b>pthread_mutexattr_setprotocol()</b>	1037

	Keyword	Name	Page
(POSIX). destroy	mutex attributes object .....	<b>pthread_mutexattr_destroy()</b>	1034
get value of protocol in	mutex attributes object/ .....	<b>pthread_mutexattr_getprotocol()</b>	1036
(POSIX). initialize	mutex attributes object .....	<b>pthread_mutexattr_init()</b>	1036
set prioceiling attr in	mutex attributes object/ .....	<b>pthread_mutexattr_setprioceiling()</b>	1037
(POSIX). initialize	mutex from attributes object.....	<b>pthread_mutex_init()</b>	1031
(POSIX). lock	mutex if it is available.....	<b>pthread_mutex_trylock()</b>	1033
destroy	mutex (POSIX). .....	<b>pthread_mutex_destroy()</b>	1030
of prioceiling attribute of	mutex (POSIX). get value.....	<b>pthread_mutex_getprioceiling()</b>	1031
lock	mutex (POSIX). .....	<b>pthread_mutex_lock()</b>	1032
set prioceiling attribute of	mutex (POSIX). dynamically ....	<b>pthread_mutex_setprioceiling()</b>	1033
unlock	mutex (POSIX). .....	<b>pthread_mutex_unlock()</b>	1034
create and initialize	mutual-exclusion semaphore. ....	<b>semMCreate()</b>	1184
library.	mutual-exclusion semaphore .....	<b>semMLib</b>	268
without restrictions. give	mutual-exclusion semaphore .....	<b>semMGiveForce()</b>	1185
between BSD IP protocol and	MUX. interface .....	<b>ipProto</b>	128
device is already loaded into	MUX. tests whether.....	<b>muxDevExists()</b>	876
load driver into	MUX. ....	<b>muxDevLoad()</b>	876
unloads device from	MUX. ....	<b>muxDevUnload()</b>	879
initialize global state for	MUX. ....	<b>muxLibInit()</b>	881
of devices registered with	MUX. display configuration .....	<b>muxShow()</b>	893
	MUX network interface library. ....	<b>muxLib</b>	186
send control information to	MUX or to device. ....	<b>muxIoctl()</b>	880
initialize and start	MUX poll task. ....	<b>muxPollStart()</b>	891
Library.	MUX toolkit Network Interface .....	<b>muxTkLib</b>	188
multicast address table from	MUX/Driver. get .....	<b>muxMCastAddrGet()</b>	884
now deprecated, see	<b>muxTkPollReceive()</b> . ....	<b>muxPollReceive()</b>	889
now deprecated, see	<b>muxTkPollSend()</b> . ....	<b>muxPollSend()</b>	890
	mv file into other directory. ....	<b>mv()</b>	905
look up symbol by	name. ....	<b>symFindByName()</b>	1291
task ID associated with task	name. look up .....	<b>taskNameToId()</b>	1337
find device using string	name. ....	<b>endFindByName()</b>	569
get socket	name. ....	<b>getsockname()</b>	638
look up host in host table by	name. ....	<b>hostGetByName()</b>	654
index given interface	name. returns interface .....	<b>ifNameToIfIndex()</b>	670
find module by	name. ....	<b>moduleFindByName()</b>	831
get current user	name and password. ....	<b>remCurIdGet()</b>	1070
set remote user	name and password. ....	<b>remCurIdSet()</b>	1070
set remote user	name and password. ....	<b>iam()</b>	657
table. verify user	name and password in login .....	<b>loginUserVerify()</b>	756
object module by specifying	name and path. unload.....	<b>unldByNameAndPath()</b>	1418
find module by file	name and path. ....	<b>moduleFindByNameAndPath()</b>	832
look up symbol by	name and type. ....	<b>symFindByNameAndType()</b>	1292
generate temporary file	name (ANSI). ....	<b>tmpnam()</b>	1391
open file specified by	name (ANSI). ....	<b>fopen()</b>	599
open file specified by	name (ANSI). ....	<b>freopen()</b>	612
ID. get	name associated with module .....	<b>moduleNameGet()</b>	834
get	name associated with task ID. ....	<b>taskName()</b>	1336
display distributed	name database filtered by type/ .....	<b>distNameFilterShow()</b>	552
(VxFusion). distributed	name database library .....	<b>distNameLib</b>	84
shared memory objects	name database library (VxMP).....	<b>smNameLib</b>	284

	Keyword	Name	Page
(VxFusion). distributed	name database show routines.....	<b>distNameShow</b>	85
(VxMP). shared memory objects	name database show routines.....	<b>smNameShow</b>	286
add entry to distributed	name database (VxFusion). .....	<b>distNameAdd()</b>	551
remove entry from distributed	name database (VxFusion). .....	<b>distNameRemove()</b>	555
display entire distributed	name database (VxFusion). .....	<b>distNameShow()</b>	555
add name to shared memory	name database (VxMP). .....	<b>smNameAdd()</b>	1237
from shared memory objects	name database (VxMP). /object .....	<b>smNameRemove()</b>	1240
/of shared memory objects	name database (VxMP). .....	<b>smNameShow()</b>	1240
preserve extra copy of task	name events (WindView).....	<b>wvTaskNamesPreserve()</b>	1484
upload preserved task	name events (WindView).....	<b>wvTaskNamesUpload()</b>	1485
expand DNS compressed	name from DNS packet.....	<b>resolvDNExpand()</b>	1074
returns interface	name given interface index. ....	<b>ifIndexToIfName()</b>	667
compress DNS	name in DNS packet.....	<b>resolvDNComp()</b>	1074
(VxFusion). find object by	name in local database.....	<b>distNameFind()</b>	553
object. set	name in thread attribute.....	<b>pthread_attr_setname()</b>	1015
descriptor/ convert device	name into DOS volume.....	<b>dosFsVolDescGet()</b>	561
get	name of connected peer.....	<b>getpeername()</b>	637
return model	name of CPU board. ....	<b>sysModel()</b>	1308
change	name of file.....	<b>rename()</b>	1072
query DNS server for host	name of IP address.....	<b>resolvGetHostByAddr()</b>	1075
type (VxFusion). look up	name of object by value and..	<b>distNameFindByValueAndType()</b>	554
set symbolic	name of this machine. ....	<b>sethostname()</b>	1201
get symbolic	name of this machine. ....	<b>gethostname()</b>	637
object. get	name of thread attribute.....	<b>pthread_attr_getname()</b>	1009
/by specifying file	name or module ID.....	<b>unld()</b>	1416
pointer. map interface	name to interface structure.....	<b>ifunit()</b>	671
database (VxMP). add	name to shared memory name .....	<b>smNameAdd()</b>	1237
bind	name to socket.....	<b>bind()</b>	430
up shared memory object by	name (VxMP). look.....	<b>smNameFind()</b>	1238
tar format. archive	named file/dir onto tape in.....	<b>tarArchive()</b>	1322
close	named semaphore (POSIX). .....	<b>sem_close()</b>	1188
initialize/open	named semaphore (POSIX). .....	<b>sem_open()</b>	1191
remove	named semaphore (POSIX). .....	<b>sem_unlink()</b>	1194
display list of fd	names in system. ....	<b>iosFdShow()</b>	711
delay for specified number of	nanoseconds. ....	<b>sysNanoDelay()</b>	1308
initiate or continue	negotiating transfer/ .....	<b>scsiSyncXferNegotiate()</b>	1162
initiate or continue	negotiating wide parameters. ....	<b>scsiWideXferNegotiate()</b>	1167
get current interrupt	nesting depth.....	<b>intCount()</b>	686
address. extract	net mask field from Internet.....	<b>bootNetmaskExtract()</b>	434
initialize	netBufLib.....	<b>netBufLibInit()</b>	907
initialize	netBufLib-managed memory pool. ....	<b>netPoolInit()</b>	923
set debug level of	<b>netDrv</b> library routines.....	<b>netDrvDebugLevelSet()</b>	913
kernel heap version of	<b>netPoolInit()</b> .....	<b>netPoolKheapInit()</b>	927
create proxy ARP	network. ....	<b>proxyNetCreate()</b>	1002
delete proxy	network. ....	<b>proxyNetDelete()</b>	1002
route to destination that is	network. add.....	<b>routeNetAdd()</b>	1120
about shared memory	network. show information.....	<b>smNetShow()</b>	1241
notation, store in/ convert	network address from dot .....	<b>inet_aton()</b>	675
notation. extract	network address in dot .....	<b>inet_netof_string()</b>	678
notation, store it in/ convert	network address to dot .....	<b>inet_ntoa_b()</b>	680

	Keyword	Name	Page
decimal notation. convert	network address to dotted .....	<b>inet_ntoa()</b>	679
form Internet address from	network and ((host number))s.....	<b>inet_makeaddr()</b>	676
form Internet address from	network and ((host number))s.....	<b>inet_makeaddr_b()</b>	677
/interface to shared memory	network (backplane) driver.....	<b>smNetLib</b>	287
	network buffer library.....	<b>netBufLib</b>	189
parameters with/ obtain set of	network configuration.....	<b>dhcpcBind()</b>	520
shut down	network connection.....	<b>shutdown()</b>	1216
control to boot ROMs. reset	network devices and transfer .....	<b>reboot()</b>	1066
shared memory	network driver show routines.....	<b>smNetShow</b>	288
deactivate specific	network events.....	<b>wvNetEventDisable()</b>	1477
activate specific	network events.....	<b>wvNetEventEnable()</b>	1478
end reporting of	network events to WindView.....	<b>wvNetDisable()</b>	1476
begin reporting	network events to WindView.....	<b>wvNetEnable()</b>	1476
priority level. enable	network events with specific .....	<b>wvNetLevelAdd()</b>	1479
priority level. disable	network events with specific .....	<b>wvNetLevelRemove()</b>	1479
driver.	Network File System (NFS) I/O .....	<b>nfsDrv</b>	195
library.	Network File System (NFS) .....	<b>nfsLib</b>	197
server library.	Network File System (NFS) .....	<b>nfsdLib</b>	193
initialize	network host table.....	<b>hostTblInit()</b>	655
routines.	network information display.....	<b>netShow</b>	192
add interface address for	network interface.....	<b>ifAddrAdd()</b>	658
delete interface address for	network interface.....	<b>ifAddrDelete()</b>	659
get Internet address of	network interface.....	<b>ifAddrGet()</b>	659
set interface address for	network interface.....	<b>ifAddrSet()</b>	660
all routes associated with	network interface. delete.....	<b>ifAllRoutesDelete()</b>	660
get broadcast address for	network interface.....	<b>ifBroadcastGet()</b>	661
set broadcast address for	network interface.....	<b>ifBroadcastSet()</b>	661
specify flags for	network interface.....	<b>ifFlagSet()</b>	664
get subnet mask for	network interface.....	<b>ifMaskGet()</b>	668
define subnet for	network interface.....	<b>ifMaskSet()</b>	668
get metric for	network interface.....	<b>ifMetricGet()</b>	669
delete routes associated with	network interface.....	<b>ifRouteDelete()</b>	670
send packet out on	network interface.....	<b>muxSend()</b>	892
packet out on Toolkit or END	network interface. send .....	<b>muxTkSend()</b>	902
delete PPP	network interface.....	<b>pppDelete()</b>	982
initialize PPP	network interface.....	<b>pppInit()</b>	985
lease request. assign	network interface and setup.....	<b>dhcpcInit()</b>	527
change	network interface flags.....	<b>ifFlagChange()</b>	663
get	network interface flags.....	<b>ifFlagGet()</b>	664
specify	network interface hop count.....	<b>ifMetricSet()</b>	669
	network interface library.....	<b>ifLib</b>	120
MUX	network interface library.....	<b>muxLib</b>	186
MUX toolkit	Network Interface Library.....	<b>muxTkLib</b>	188
	network interface library.....	<b>netLib</b>	192
display attached	network interfaces.....	<b>ifShow()</b>	671
address. return	network number from Internet .....	<b>inet_netof()</b>	678
address. convert Internet	network number from string to.....	<b>inet_network()</b>	679
initialize	network package.....	<b>netLibInit()</b>	915
install	network remote file driver.....	<b>netDrv()</b>	913
driver.	network remote file I/O .....	<b>netDrv</b>	190

	Keyword	Name	Page
library.	network route manipulation .....	<b>routeLib</b>	238
print synopsis of	network routines.....	<b>netHelp()</b>	914
create binding between	network service and END. ....	<b>muxBind()</b>	874
device.	network service from specified.....	<b>muxUnbind()</b>	904
initialize	network show routines. ....	<b>netShowInit()</b>	928
attach routine for TCP/IP	network stack. generic.....	<b>ipAttach()</b>	714
detach routine for TCP/IP	network stack. generic.....	<b>ipDetach()</b>	714
statistics. show	network stack data pool.....	<b>netStackDataPoolShow()</b>	929
statistics. show	network stack system pool .....	<b>netStackSysPoolShow()</b>	929
	network task entry point. ....	<b>netTask()</b>	930
client library. Simple	Network Time Protocol (SNTP).....	<b>sntpcLib</b>	291
server library. Simple	Network Time Protocol (SNTP).....	<b>sntpsLib</b>	292
time. initialize	network with DHCP at boot .....	<b>dhcpcBootBind()</b>	521
WindView for	Networking Interface Library. ....	<b>wvNetLib</b>	356
show proxy ARP	networks.....	<b>proxyNetShow()</b>	1003
function (C++). set	new_handler to user-defined .....	<b>set_new_handler()</b>	1199
information from requested	NFS device. /configuration.....	<b>nfsDevInfoGet()</b>	934
unmount	NFS device. ....	<b>nfsUnmount()</b>	943
display mounted	NFS devices. ....	<b>nfsDevShow()</b>	935
create list of all	NFS devices in system.....	<b>nfsDevListGet()</b>	935
install	NFS driver. ....	<b>nfsDrv()</b>	937
IO system driver number for	NFS driver. return .....	<b>nfsDrvNumGet()</b>	937
specify file system to be	NFS exported. ....	<b>nfsExport()</b>	939
mount	NFS file system. ....	<b>nfsMount()</b>	941
display	NFS help menu.....	<b>nfsHelp()</b>	940
Network File System	(NFS) I/O driver. ....	<b>nfsDrv</b>	195
Network File System	(NFS) library.....	<b>nfsLib</b>	197
initialize	NFS server. ....	<b>nfsdInit()</b>	936
get status of	NFS server. ....	<b>nfsdStatusGet()</b>	938
show status of	NFS server. ....	<b>nfsdStatusShow()</b>	938
Network File System	(NFS) server library.....	<b>nfsdLib</b>	193
parameters. get	NFS UNIX authentication.....	<b>nfsAuthUnixGet()</b>	932
parameters. modify	NFS UNIX authentication.....	<b>nfsAuthUnixPrompt()</b>	933
parameters. set	NFS UNIX authentication.....	<b>nfsAuthUnixSet()</b>	933
parameters. display	NFS UNIX authentication.....	<b>nfsAuthUnixShow()</b>	934
parameters. set ID number of	NFS UNIX authentication.....	<b>nfsIdSet()</b>	941
get type of <b>select()</b> wake-up	node. ....	<b>selWakeupType()</b>	1174
node in list after specified	node. insert .....	<b>lstInsert()</b>	766
steps away from specified	node. find list node nStep .....	<b>lstNStep()</b>	768
remove hash	node from hash table. ....	<b>hashTblRemove()</b>	650
delete specified	node from list.....	<b>lstDelete()</b>	763
delete and return first	node from list.....	<b>lstGet()</b>	765
wake-up list. find and delete	node from <b>select()</b> .....	<b>selNodeDelete()</b>	1171
call routine for each	node in hash table. ....	<b>hashTblEach()</b>	648
find	node in list.....	<b>lstFind()</b>	764
find first	node in list.....	<b>lstFirst()</b>	764
find last	node in list.....	<b>lstLast()</b>	767
find next	node in list.....	<b>lstNext()</b>	768
find Nth	node in list.....	<b>lstNth()</b>	769
find previous	node in list.....	<b>lstPrevious()</b>	769

	Keyword	Name	Page
	node. insert	node in list after specified .....	<b>lstInsert()</b> 766
	table. put hash	node into specified hash .....	<b>hashTblPut()</b> 649
	specified node. find list	node nStep steps away from .....	<b>lstNStep()</b> 768
	key. find hash	node that matches specified .....	<b>hashTblFind()</b> 648
	add	node to end of list .....	<b>lstAdd()</b> 761
	add wake-up	node to <b>select()</b> wake-up list .....	<b>selNodeAdd()</b> 1171
	and bootstrap current	node (VxFusion). initialize .....	<b>distInit()</b> 549
	report number of	nodes in list. ....	<b>lstCount()</b> 762
	wake-up list. get number of	nodes in <b>select()</b> .....	<b>selWakeupListLen()</b> 1173
	update contents of interface	non-counter object .....	<b>m2IfVariableUpdate()</b> 785
	saved environment/ perform	non-local goto by restoring .....	<b>longjmp()</b> 759
	(ANSI). compute	non-negative square root .....	<b>sqrt()</b> 1261
	(ANSI). compute	non-negative square root .....	<b>sqrtf()</b> 1261
	add hook to install static and	non-RIP routes into RIP. ....	<b>ripRouteHookAdd()</b> 1099
	get contents of	non-volatile RAM. ....	<b>sysNvRamGet()</b> 1309
	write to	non-volatile RAM. ....	<b>sysNvRamSet()</b> 1310
/whether character is printing,	non-white-space character/ .....	<b>isgraph()</b> 722	
support for operator new	(nothrow) (C++). /run-time .....	<b>operator new()</b> 947	
message queue. start event	notification process for .....	<b>msgQEvStart()</b> 859	
message queue. stop event	notification process for .....	<b>msgQEvStop()</b> 860	
semaphore. start event	notification process for .....	<b>semEvStart()</b> 1179	
semaphore. stop event	notification process for .....	<b>semEvStop()</b> 1181	
receive packet from	NPT driver .....	<b>muxTkReceive()</b> 901	
out in polled mode to END or	NPT interface. send packet .....	<b>muxTkPollSend()</b> 899	
poll for packet from	NPT or END driver. ....	<b>muxTkPollReceive()</b> 898	
checks if device is	NPT or END interface .....	<b>muxTkDrvCheck()</b> 898	
bind	NPT protocol to driver .....	<b>muxTkBind()</b> 895	
specified/ find list node	nStep steps away from .....	<b>lstNStep()</b> 768	
pass-through (to Windows	NT) file system library. ....	<b>ntPassFsLib</b> 198	
convert portions of second to	NTP format .....	<b>sntpsNsecToFraction()</b> 1251	
associate device with	ntPassFs file system/ .....	<b>ntPassFsDevInit()</b> 943	
prepare to use	ntPassFs library .....	<b>ntPassFsInit()</b> 944	
reload	object module .....	<b>reld()</b> 1069	
get information about	object module .....	<b>moduleInfoGet()</b> 833	
file name or module/ unload	object module by specifying .....	<b>unld()</b> 1416	
group number. unload	object module by specifying .....	<b>unldByGroup()</b> 1417	
module ID. unload	object module by specifying .....	<b>unldByModuleId()</b> 1417	
name and path. unload	object module by specifying .....	<b>unldByNameAndPath()</b> 1418	
load	object module into memory. ....	<b>ld()</b> 730	
load	object module into memory. ....	<b>loadModule()</b> 739	
load	object module into memory. ....	<b>loadModuleAt()</b> 739	
	object module loader .....	<b>loadLib</b> 133	
library.	object module management .....	<b>moduleLib</b> 176	
library.	object module unloading .....	<b>unldLib</b> 334	
initialize	on-board SCSI port .....	<b>sysScsiInit()</b> 1314	
(POSIX).	open directory for searching .....	<b>opendir()</b> 946	
driver-specific/ validate	open fd and return .....	<b>iosFdValue()</b> 711	
	open file. ....	<b>open()</b> 945	
(POSIX).	open file specified by fd .....	<b>fdopen()</b> 587	
(ANSI).	open file specified by name .....	<b>fopen()</b> 599	

	Keyword	Name	Page
	(ANSI). open file specified by name .....	<b>freopen()</b>	612
	open message queue (POSIX). .....	<b>mq_open()</b>	841
	open socket. ....	<b>socket()</b>	1252
port bound to it.	open socket with privileged .....	<b>rresvport()</b>	1125
TSFS socket (Windview).	open upload path to host using .....	<b>tsfsUploadPathCreate()</b>	1402
default run-time support for	operator new (C++). .....	<b>operator new()</b>	947
default run-time support for	operator new (nothrow) (C++). .....	<b>operator new()</b>	947
(C++). run-time support for	operator new with placement .....	<b>operator new()</b>	948
invert	order of bytes in buffer. ....	<b>binvert()</b>	431
specify amount of debugging	output. ....	<b>ripDebugLevelSet()</b>	1092
interrupt-level	output. ....	<b>tyITx()</b>	1410
argument list to standard	output (ANSI). /with variable .....	<b>vprintf()</b>	1446
line buffering for standard	output or standard error. set .....	<b>setlinebuf()</b>	1202
write character to standard	output stream (ANSI). ....	<b>putchar()</b>	1046
write string to standard	output stream (ANSI). ....	<b>puts()</b>	1047
formatted string to standard	output stream (ANSI). write .....	<b>printf()</b>	997
return timer expiration	overrun (POSIX). ....	<b>timer_getoverrun()</b>	1383
get contents of	P5 PMC0. ....	<b>pentiumP5PmcGet0()</b>	958
stop	P5 PMC0. ....	<b>pentiumP5PmcStop0()</b>	962
get contents of	P5 PMC0 and PMC1. ....	<b>pentiumP5PmcGet()</b>	958
stop both	P5 PMC0 and PMC1. ....	<b>pentiumP5PmcStop()</b>	962
get contents of	P5 PMC1. ....	<b>pentiumP5PmcGet1()</b>	959
stop	P5 PMC1. ....	<b>pentiumP5PmcStop1()</b>	963
initialize RPC	package. ....	<b>rpcInit()</b>	1124
task's access to RPC	package. initialize .....	<b>rpcTaskInit()</b>	1124
task CPU utilization tool	package. initialize .....	<b>spyLibInit()</b>	1259
initialize local debugging	package. ....	<b>dbgInit()</b>	514
initialize exception handling	package. ....	<b>excInit()</b>	580
initialize network	package. ....	<b>netLibInit()</b>	915
(POSIX). dynamic	package initialization .....	<b>pthread_once()</b>	1038
/distributed message queue show	package (VxFusion). ....	<b>msgQDistShowInit()</b>	859
compress DNS name in DNS	packet. ....	<b>resolvDNComp()</b>	1074
DNS compressed name from DNS	packet. expand. ....	<b>resolvDNExpand()</b>	1074
attach link-level header to	packet. ....	<b>muxLinkHeaderCreate()</b>	881
addressing information from	packet. get .....	<b>muxPacketAddrGet()</b>	885
return data from	packet. ....	<b>muxPacketDataGet()</b>	886
install interface	packet counter routine. ....	<b>m2IfPktCountRtnInstall()</b>	779
increment interface	packet counters. ....	<b>m2IfGenericPacketCount()</b>	777
device. increment	packet counters for 802.3 .....	<b>m2If8023PacketCount()</b>	772
library. Berkeley	Packet Filter (BPF) I/O driver .....	<b>bpfdrv</b>	35
create Berkeley	Packet Filter device. ....	<b>bpfdDevCreate()</b>	442
destroy Berkeley	Packet Filter device. ....	<b>bpfdDevDelete()</b>	442
receive	packet from NPT driver. ....	<b>muxTkReceive()</b>	901
poll for	packet from NPT or END driver. ....	<b>muxTkPollReceive()</b>	898
library.	Packet InterNet Groper (PING) .....	<b>pingLib</b>	209
END or NPT interface. send	packet out in polled mode to .....	<b>muxTkPollSend()</b>	899
interface. send	packet out on network .....	<b>muxSend()</b>	892
network interface. send	packet out on Toolkit or END .....	<b>muxTkSend()</b>	902
receive all internet protocol	packets. add routine to .....	<b>ipFilterHookAdd()</b>	715
get architecture-dependent	page block size (VxVMI). ....	<b>vmPageBlockSizeGet()</b>	1441

	Keyword	Name	Page
	allocate memory on page boundary.....	<b>valloc()</b>	1429
	clear page from CY7C604 cache.....	<b>cacheCy604ClearPage()</b>	450
	clear page from Sun-4 cache.....	<b>cacheSun4ClearPage()</b>	471
(VxVMI). get state of	page of virtual memory.....	<b>vmStateGet()</b>	1443
return	page size.....	<b>vmBasePageSizeGet()</b>	1433
return	page size (VxVMI).....	<b>vmPageSizeGet()</b>	1442
lock specified	pages into memory (POSIX).....	<b>mlock()</b>	822
unlock specified	pages (POSIX).....	<b>munlock()</b>	869
shared global/ map physical	pages to virtual space in.....	<b>vmGlobalMap()</b>	1438
memory (POSIX). lock all	pages used by process into.....	<b>mlockall()</b>	823
unlock all	pages used by process (POSIX).....	<b>munlockall()</b>	869
fields in SCSI logical	partition. initialize.....	<b>scsiBlkDevInit()</b>	1138
retrieve handle for	partition.....	<b>dpartPartGet()</b>	565
add memory to system memory	partition.....	<b>memAddToPool()</b>	806
free block in system memory	partition. find largest.....	<b>memFindMax()</b>	811
options for system memory	partition. set debug.....	<b>memOptionsSet()</b>	812
add memory to memory	partition.....	<b>memPartAddToPool()</b>	813
allocate aligned memory from	partition.....	<b>memPartAlignedAlloc()</b>	813
allocate block of memory from	partition.....	<b>memPartAlloc()</b>	814
create memory	partition.....	<b>memPartCreate()</b>	814
free block of memory in	partition.....	<b>memPartFree()</b>	815
set debug options for memory	partition.....	<b>memPartOptionsSet()</b>	816
block of memory in specified	partition. reallocate.....	<b>memPartRealloc()</b>	817
of memory from system memory	partition (ANSI). /block.....	<b>malloc()</b>	802
free shared memory system	partition block of memory/.....	<b>smMemFree()</b>	1233
statistics. show	partition blocks and.....	<b>memPartShow()</b>	818
show system memory	partition blocks and/.....	<b>memShow()</b>	820
show shared memory system	partition blocks and/.....	<b>smMemShow()</b>	1236
parse and display	partition data.....	<b>usrFdiskPartShow()</b>	1423
FDISK-style	partition handler.....	<b>usrFdiskPartLib</b>	337
get	partition information.....	<b>memPartInfoGet()</b>	816
full-featured memory	partition manager.....	<b>memLib</b>	158
core memory	partition manager.....	<b>memPartLib</b>	160
generic disk	partition manager.....	<b>dpartCbio</b>	98
block device. define logical	partition on SCSI.....	<b>scsiBlkDevCreate()</b>	1137
initialize memory	partition show facility.....	<b>memShowInit()</b>	821
read FDISK-style	partition table.....	<b>usrFdiskPartRead()</b>	1422
create FDISK-like	partition table on disk.....	<b>usrFdiskPartCreate()</b>	1421
memory to shared memory system	partition (VxMP). add.....	<b>smMemAddToPool()</b>	1231
from shared memory system	partition (VxMP). /for array.....	<b>smMemCalloc()</b>	1232
block in shared memory system	partition (VxMP). /free.....	<b>smMemFindMax()</b>	1233
from shared memory system	partition (VxMP). /of memory.....	<b>smMemMalloc()</b>	1234
for shared memory system	partition (VxMP). /options.....	<b>smMemOptionsSet()</b>	1234
from shared memory system	partition (VxMP). /of memory.....	<b>smMemRealloc()</b>	1235
create shared memory	partition (VxMP).....	<b>memPartSmCreate()</b>	818
initialize	partitioned disk.....	<b>dpartDevCreate()</b>	564
into integer and fraction	parts (ANSI). /number.....	<b>modf()</b>	827
processor (Unimplemented)/	pass string to command.....	<b>system()</b>	1317
associate device with	passFs file system functions.....	<b>passFsDevInit()</b>	949
prepare to use	passFs library.....	<b>passFsInit()</b>	949

	Keyword	Name	Page
system library (VxSim).	pass-through (to UNIX) file.....	<b>passFsLib</b>	200
file system library.	pass-through (to Windows NT).....	<b>ntPassFsLib</b>	198
get current user name and	password.....	<b>remCurIdGet()</b>	1070
set remote user name and	password.....	<b>remCurIdSet()</b>	1070
set remote user name and	password.....	<b>iam()</b>	657
default	password encryption routine.....	<b>loginDefaultEncrypt()</b>	751
verify user name and	password in login table.....	<b>loginUserVerify()</b>	756
FTP data connection using	PASV mode. initialize.....	<b>ftpDataConnInitPassiveMode()</b>	624
file status information using	pathname (POSIX). get.....	<b>stat()</b>	1268
file status information using	pathname (POSIX). get.....	<b>statfs()</b>	1268
get name of connected	peer.....	<b>getpeername()</b>	637
address of point-to-point	peer. get Internet.....	<b>ifDstAddrGet()</b>	662
	pend on set of fds.....	<b>select()</b>	1169
wake up task	pending in <b>select()</b> .....	<b>selWakeup()</b>	1172
unlock every task	pending on semaphore.....	<b>semFlush()</b>	1181
delivery/ retrieve set of	pending signals blocked from.....	<b>sigpending()</b>	1221
library.	Pentium and Pentium[234].....	<b>pentiumLib</b>	205
specific show routines.	Pentium and Pentium[234].....	<b>pentiumShow</b>	209
specific routines.	Pentium and PentiumPro.....	<b>pentiumALib</b>	201
Pentium and	Pentium[234] library.....	<b>pentiumLib</b>	205
routines. Pentium and	Pentium[234] specific show.....	<b>pentiumShow</b>	209
Pentium and	PentiumPro specific routines.....	<b>pentiumALib</b>	201
MMU library for	PentiumPro/2/3/4 32 bit mode.....	<b>mmuPro32Lib</b>	163
Counters). show PMCs	(Performance Monitoring.....	<b>pentiumPmcShow()</b>	970
reports. begin	periodic task activity.....	<b>spy()</b>	1257
reports. run	periodic task activity.....	<b>spyTask()</b>	1260
spawn task to call function	periodically.....	<b>period()</b>	975
call function	periodically.....	<b>periodRun()</b>	976
configure SCSI	peripherals.....	<b>usrScsiConfig()</b>	1425
for server. assign	permanent address storage hook.....	<b>dhcpsAddressHookAdd()</b>	538
for server. assign	permanent lease storage hook.....	<b>dhcpsLeaseHookAdd()</b>	541
translate virtual address to	physical address (ARM).....	<b>mmuVirtToPhys()</b>	826
translate	physical address for drivers.....	<b>cacheDrvPhysToVirt()</b>	455
translate	physical address for drivers.....	<b>cacheTiTms390PhysToVirt()</b>	474
list. add	physical address into linked.....	<b>rcvEtherAddrAdd()</b>	1058
address (ARM). translate	physical address to virtual.....	<b>mmuPhysToVirt()</b>	823
translate virtual address to	physical address (VxVMI).....	<b>vmTranslate()</b>	1445
structures on specified	physical device. show BLK_DEV.....	<b>scsiBlkDevShow()</b>	1139
show status information for	physical device.....	<b>scsiPhysDevShow()</b>	1153
create SCSI	physical device structure.....	<b>scsiPhysDevCreate()</b>	1151
SCSI controller. list	physical devices attached to.....	<b>scsiShow()</b>	1160
space in shared global/ map	physical pages to virtual.....	<b>vmGlobalMap()</b>	1438
move tape on specified	physical SCSI device.....	<b>scsiSpace()</b>	1161
space (VxVMI). map	physical space into virtual.....	<b>vmMap()</b>	1440
delete SCSI	physical-device structure.....	<b>scsiPhysDevDelete()</b>	1152
Packet InterNet Groper	(PING) library.....	<b>pingLib</b>	209
initialize	<b>ping()</b> utility.....	<b>pingLibInit()</b>	978
create	pipe device.....	<b>pipeDevCreate()</b>	979
delete	pipe device.....	<b>pipeDevDelete()</b>	979
initialize	pipe driver.....	<b>pipeDrv()</b>	980

	Keyword	Name	Page
	pipe I/O driver.....	pipeDrv	210
support for operator new with	placement (C++).. run-time .....	operator new()	948
get contents of P5	PMC0.....	pentiumP5PmcGet0()	958
reset	PMC0.....	pentiumP5PmcReset0()	960
start	PMC0.....	pentiumP5PmcStart0()	961
stop P5	PMC0.....	pentiumP5PmcStop0()	962
get contents of	PMC0.....	pentiumP6PmcGet0()	964
reset	PMC0.....	pentiumP6PmcReset0()	965
get contents of	PMC0.....	pentiumPmcGet0()	968
reset	PMC0.....	pentiumPmcReset0()	969
start	PMC0.....	pentiumPmcStart0()	971
stop	PMC0.....	pentiumPmcStop0()	972
get contents of P5	PMC0 and PMC1.....	pentiumP5PmcGet()	958
reset both	PMC0 and PMC1.....	pentiumP5PmcReset()	959
stop both P5	PMC0 and PMC1.....	pentiumP5PmcStop()	962
get contents of	PMC0 and PMC1.....	pentiumP6PmcGet()	963
reset both	PMC0 and PMC1.....	pentiumP6PmcReset()	965
start both	PMC0 and PMC1.....	pentiumP6PmcStart()	966
stop both	PMC0 and PMC1.....	pentiumP6PmcStop()	966
get contents of	PMC0 and PMC1.....	pentiumPmcGet()	967
reset both	PMC0 and PMC1.....	pentiumPmcReset()	969
start both	PMC0 and PMC1.....	pentiumPmcStart()	970
stop both	PMC0 and PMC1.....	pentiumPmcStop()	972
get contents of P5 PMC0 and	PMC1.....	pentiumP5PmcGet()	958
get contents of P5	PMC1.....	pentiumP5PmcGet1()	959
reset both PMC0 and	PMC1.....	pentiumP5PmcReset()	959
reset	PMC1.....	pentiumP5PmcReset1()	960
start	PMC1.....	pentiumP5PmcStart1()	961
stop both P5 PMC0 and	PMC1.....	pentiumP5PmcStop()	962
stop P5	PMC1.....	pentiumP5PmcStop1()	963
get contents of PMC0 and	PMC1.....	pentiumP6PmcGet()	963
get contents of	PMC1.....	pentiumP6PmcGet1()	964
reset both PMC0 and	PMC1.....	pentiumP6PmcReset()	965
reset	PMC1.....	pentiumP6PmcReset1()	965
start both PMC0 and	PMC1.....	pentiumP6PmcStart()	966
stop both PMC0 and	PMC1.....	pentiumP6PmcStop()	966
stop	PMC1.....	pentiumP6PmcStop1()	967
get contents of PMC0 and	PMC1.....	pentiumPmcGet()	967
get contents of	PMC1.....	pentiumPmcGet1()	968
reset both PMC0 and	PMC1.....	pentiumPmcReset()	969
reset	PMC1.....	pentiumPmcReset1()	969
start both PMC0 and	PMC1.....	pentiumPmcStart()	970
start	PMC1.....	pentiumPmcStart1()	971
stop both PMC0 and	PMC1.....	pentiumPmcStop()	972
stop	PMC1.....	pentiumPmcStop1()	972
Counters). show	PMCs (Performance Monitoring.....	pentiumPmcShow()	970
into DOS volume descriptor	pointer. convert device name.....	dosFs VolDescGet()	561
name to interface structure	pointer. map interface.....	ifunit()	671
set file read/write	pointer.....	lseek()	760
test error indicator for file	pointer (ANSI).....	ferror()	589

	Keyword	Name	Page
	advance ring pointer by n bytes. ....	<b>rngMoveAhead()</b>	1112
	display file pointer internals. ....	<b>stdioShow()</b>	1270
	structure. return pointer to SCSI_PHYS_DEV. ....	<b>scsiPhysDevIdGet()</b>	1152
buffer without moving ring	pointers. /byte ahead in ring. ....	<b>rngPutAhead()</b>	1113
extract socket address	pointers from route message. ....	<b>ripAddrXtract()</b>	1083
address for other end of	point-to-point link. define. ....	<b>ifDstAddrSet()</b>	663
get Internet address of	point-to-point peer. ....	<b>ifDstAddrGet()</b>	662
library.	Point-to-Point Protocol. ....	<b>pppLib</b>	212
routines.	Point-to-Point Protocol show. ....	<b>pppShow</b>	215
parameters/ set scheduling	policy and scheduling. ....	<b>sched_setscheduler()</b>	1135
get current scheduling	policy (POSIX). ....	<b>sched_getscheduler()</b>	1133
END driver.	poll for packet from NPT or. ....	<b>muxTkPollReceive()</b>	898
initialize and start MUX	poll task. ....	<b>muxPollStart()</b>	891
adds device to list	polled by tMuxPollTask. ....	<b>muxPollDevAdd()</b>	887
removes device from list	polled by tMuxPollTask. ....	<b>muxPollDevDel()</b>	887
/whether device is on list	polled by tMuxPollTask. ....	<b>muxPollDevStat()</b>	888
interface. send packet out in	polled mode to END or NPT. ....	<b>muxTkPollSend()</b>	899
get delay on	polling task. ....	<b>muxTaskDelayGet()</b>	893
set inter-cycle delay on	polling task. ....	<b>muxTaskDelaySet()</b>	894
add another entry to address	pool. ....	<b>dhcpsLeaseEntryAdd()</b>	540
construct back to memory	pool. free cBlk-cluster. ....	<b>netCIBlkFree()</b>	907
free cluster back to memory	pool. ....	<b>netCIFree()</b>	909
cluster from specified cluster	pool. get. ....	<b>netClusterGet()</b>	911
free mBlk back to memory	pool. ....	<b>netMblkFree()</b>	921
get mBlk from memory	pool. ....	<b>netMblkGet()</b>	921
delete memory	pool. ....	<b>netPoolDelete()</b>	923
netBufLib-managed memory	pool. initialize. ....	<b>netPoolInit()</b>	923
allocate telegram buffer from	pool of buffers (VxFusion). ....	<b>distTBufAlloc()</b>	556
return telegram buffer to	pool of buffers (VxFusion). ....	<b>distTBufFree()</b>	557
show	pool statistics. ....	<b>netPoolShow()</b>	927
show network stack data	pool statistics. ....	<b>netStackDataPoolShow()</b>	929
show network stack system	pool statistics. ....	<b>netStackSysPoolShow()</b>	929
stack (POSIX).	pop cleanup routine off top of. ....	<b>pthread_cleanup_pop()</b>	1019
forwarding for particular	port. disable broadcast. ....	<b>proxyPortFwdOff()</b>	1003
forwarding for particular	port. enable broadcast. ....	<b>proxyPortFwdOn()</b>	1004
initialize on-board SCSI	port. ....	<b>sysScsiInit()</b>	1314
bind socket to privileged IP	port. ....	<b>bindresvport()</b>	431
open socket with privileged	port bound to it. ....	<b>rresvport()</b>	1125
FTP data connection using	PORT mode. initialize. ....	<b>ftpDataConnInit()</b>	623
remove	port number filter for events. ....	<b>wvNetPortFilterClear()</b>	1480
forwarding. show	ports enabled for broadcast. ....	<b>proxyPortShow()</b>	1004
store current value of file	position indicator for stream/. ....	<b>fgetpos()</b>	590
(ANSI). set file	position indicator for stream. ....	<b>fseek()</b>	618
(ANSI). set file	position indicator for stream. ....	<b>fsetpos()</b>	619
return current value of file	position indicator for stream/. ....	<b>ftell()</b>	620
beginning of file/ set file	position indicator to. ....	<b>rewind()</b>	1082
(POSIX). reset	position to start of directory. ....	<b>rewinddir()</b>	1082
interfaces.	POSIX 1003.1c thread library. ....	<b>pthreadLib</b>	217
thread attributes object	(POSIX). destroy. ....	<b>pthread_attr_destroy()</b>	1007
in thread attributes object	(POSIX). /attribute. ....	<b>pthread_attr_getdetachstate()</b>	1008

	Keyword	Name	Page
in thread attributes object	(POSIX). /attribute.....	<b>pthread_attr_getinheritsched()</b>	1008
in thread attributes object	(POSIX). /schedparam attribute.	<b>pthread_attr_getschedparam()</b>	1009
from thread attributes object	(POSIX). /attribute.....	<b>pthread_attr_getschedpolicy()</b>	1010
scope from thread attributes	(POSIX). get contention.....	<b>pthread_attr_getscope()</b>	1011
from thread attributes object	(POSIX). /stackaddr attribute.....	<b>pthread_attr_getstackaddr()</b>	1011
in thread attributes object	(POSIX). /stacksize attribute.....	<b>pthread_attr_getstacksize()</b>	1012
thread attributes object	(POSIX). initialize.....	<b>pthread_attr_init()</b>	1012
in thread attributes object	(POSIX). /attribute.....	<b>pthread_attr_setdetachstate()</b>	1013
in thread attribute object	(POSIX). /attribute.....	<b>pthread_attr_setinheritsched()</b>	1014
in thread attributes object	(POSIX). /schedparam attribute.	<b>pthread_attr_setschedparam()</b>	1015
in thread attributes object	(POSIX). /attribute.....	<b>pthread_attr_setschedpolicy()</b>	1016
scope for thread attributes	(POSIX). set contention.....	<b>pthread_attr_setscope()</b>	1017
in thread attributes object	(POSIX). /stackaddr attribute.....	<b>pthread_attr_setstackaddr()</b>	1017
in thread attributes object	(POSIX). /stacksize attribute.....	<b>pthread_attr_setstacksize()</b>	1018
cancel execution of thread	(POSIX). .....	<b>pthread_cancel()</b>	1018
routine off top of stack	(POSIX). pop cleanup.....	<b>pthread_cleanup_pop()</b>	1019
routine onto cleanup stack	(POSIX). pushes.....	<b>pthread_cleanup_push()</b>	1019
threads waiting on condition	(POSIX). unblock all.....	<b>pthread_cond_broadcast()</b>	1020
destroy condition variable	(POSIX). .....	<b>pthread_cond_destroy()</b>	1020
initialize condition variable	(POSIX). .....	<b>pthread_cond_init()</b>	1021
thread waiting on condition	(POSIX). unblock.....	<b>pthread_cond_signal()</b>	1022
variable with timeout	(POSIX). wait for condition.....	<b>pthread_cond_timedwait()</b>	1022
wait for condition variable	(POSIX). .....	<b>pthread_cond_wait()</b>	1023
condition attributes object	(POSIX). destroy.....	<b>pthread_condattr_destroy()</b>	1024
condition attribute object	(POSIX). initialize.....	<b>pthread_condattr_init()</b>	1024
create thread	(POSIX). .....	<b>pthread_create()</b>	1025
dynamically detach thread	(POSIX). .....	<b>pthread_detach()</b>	1025
compare thread IDs	(POSIX). .....	<b>pthread_equal()</b>	1026
terminate thread	(POSIX). .....	<b>pthread_exit()</b>	1026
attribute from thread	(POSIX). /value of schedparam.....	<b>pthread_getschedparam()</b>	1027
get thread specific data	(POSIX). .....	<b>pthread_getspecific()</b>	1027
wait for thread to terminate	(POSIX). .....	<b>pthread_join()</b>	1028
thread specific data key	(POSIX). create.....	<b>pthread_key_create()</b>	1029
thread specific data key	(POSIX). delete.....	<b>pthread_key_delete()</b>	1029
send signal to thread	(POSIX). .....	<b>pthread_kill()</b>	1030
destroy mutex	(POSIX). .....	<b>pthread_mutex_destroy()</b>	1030
prioceiling attribute of mutex	(POSIX). get value of.....	<b>pthread_mutex_getprioceiling()</b>	1031
mutex from attributes object	(POSIX). initialize.....	<b>pthread_mutex_init()</b>	1031
lock mutex	(POSIX). .....	<b>pthread_mutex_lock()</b>	1032
prioceiling attribute of mutex	(POSIX). dynamically set.....	<b>pthread_mutex_setprioceiling()</b>	1033
lock mutex if it is available	(POSIX). .....	<b>pthread_mutex_trylock()</b>	1033
mutex attributes object	(POSIX). destroy.....	<b>pthread_mutexattr_destroy()</b>	1034
unlock mutex	(POSIX). .....	<b>pthread_mutex_unlock()</b>	1034
prioceiling attr in mutex attr object	(POSIX). /value of.....	<b>pthread_mutexattr_getprioceiling()</b>	1035
in mutex attributes object	(POSIX). /value of protocol....	<b>pthread_mutexattr_getprotocol()</b>	1036
mutex attributes object	(POSIX). initialize.....	<b>pthread_mutexattr_init()</b>	1036
in mutex attributes object	(POSIX). set prioceiling attr	<b>pthread_mutexattr_setprioceiling()</b>	1037
in mutex attribute object	(POSIX). /protocol attribute....	<b>pthread_mutexattr_setprotocol()</b>	1037
dynamic package initialization	(POSIX). .....	<b>pthread_once()</b>	1038
get calling thread's ID	(POSIX). .....	<b>pthread_self()</b>	1039

	Keyword	Name	Page
state for calling thread	(POSIX). set cancellation.....	<b>pthread_setcancelstate()</b>	1039
type for calling thread	(POSIX). set cancellation.....	<b>pthread_setcanceltype()</b>	1040
attribute for thread	(POSIX). /set schedparam.....	<b>pthread_setschedparam()</b>	1040
set thread specific data	(POSIX).....	<b>pthread_setspecific()</b>	1041
calling thread's signal mask	(POSIX). /and/or examine.....	<b>pthread_sigmask()</b>	1042
point in calling thread	(POSIX). create cancellation.....	<b>pthread_testcancel()</b>	1043
read one entry from directory	(POSIX).....	<b>readdir()</b>	1064
position to start of directory	(POSIX). reset.....	<b>rewinddir()</b>	1082
get maximum priority	(POSIX).....	<b>sched_get_priority_max()</b>	1131
parameters for specified task	(POSIX). get scheduling.....	<b>sched_getparam()</b>	1132
get minimum priority	(POSIX).....	<b>sched_get_priority_min()</b>	1132
get current scheduling policy	(POSIX).....	<b>sched_getscheduler()</b>	1133
get current time slice	(POSIX).....	<b>sched_rr_get_interval()</b>	1134
set task's priority	(POSIX).....	<b>sched_setparam()</b>	1134
and scheduling parameters	(POSIX). /scheduling policy.....	<b>sched_setscheduler()</b>	1135
relinquish CPU	(POSIX).....	<b>sched_yield()</b>	1136
close named semaphore	(POSIX).....	<b>sem_close()</b>	1188
destroy unnamed semaphore	(POSIX).....	<b>sem_destroy()</b>	1189
get value of semaphore	(POSIX).....	<b>sem_getvalue()</b>	1190
initialize unnamed semaphore	(POSIX).....	<b>sem_init()</b>	1191
named semaphore	(POSIX). initialize/open.....	<b>sem_open()</b>	1191
unlock (give) semaphore	(POSIX).....	<b>sem_post()</b>	1193
returning error if unavailable	(POSIX). /(take) semaphore,.....	<b>sem_trywait()</b>	1194
remove named semaphore	(POSIX).....	<b>sem_unlink()</b>	1194
blocking if not available	(POSIX). /(take) semaphore,.....	<b>sem_wait()</b>	1195
action associated with signal	(POSIX). /and/or specify.....	<b>sigaction()</b>	1216
add signal to signal set	(POSIX).....	<b>sigaddset()</b>	1217
delete signal from signal set	(POSIX).....	<b>sigdelset()</b>	1218
set with no signals included	(POSIX). initialize signal.....	<b>sigemptyset()</b>	1218
set with all signals included	(POSIX). initialize signal.....	<b>sigfillset()</b>	1219
see if signal is in signal set	(POSIX). test to.....	<b>sigismember()</b>	1220
signals blocked from delivery	(POSIX). /set of pending.....	<b>sigpending()</b>	1221
and/or change signal mask	(POSIX). examine.....	<b>sigprocmask()</b>	1221
task until delivery of signal	(POSIX). suspend.....	<b>sigsuspend()</b>	1224
for signal to be delivered	(POSIX). wait.....	<b>sigwait()</b>	1226
information using pathname	(POSIX). get file status.....	<b>stat()</b>	1268
information using pathname	(POSIX). get file status.....	<b>statfs()</b>	1268
error number to error string	(POSIX). map.....	<b>strerror_r()</b>	1274
string into tokens (reentrant)	(POSIX). break down.....	<b>strtok_r()</b>	1283
clock for timing base	(POSIX). /using specified.....	<b>timer_create()</b>	1382
previously created timer	(POSIX). remove.....	<b>timer_delete()</b>	1383
timer expiration overrun	(POSIX). return.....	<b>timer_getoverrun()</b>	1383
expiration and reload value	(POSIX). /time before.....	<b>timer_gettime()</b>	1384
next expiration and arm timer	(POSIX). set time until.....	<b>timer_settime()</b>	1384
delete file	(POSIX).....	<b>unlink()</b>	1418
memory management library	(POSIX).....	<b>mmanPxLib</b>	162
message queue library	(POSIX).....	<b>mqPxLib</b>	179
scheduling library	(POSIX).....	<b>schedPxLib</b>	245
synchronization library	(POSIX). semaphore.....	<b>semPxLib</b>	271
timer library	(POSIX).....	<b>timerLib</b>	322

	Keyword	Name	Page
of asynchronous I/O operation	(POSIX). /error status.....	<b>aio_error()</b>	408
initiate asynchronous read	(POSIX).....	<b>aio_read()</b>	408
of asynchronous I/O operation	(POSIX). /return status .....	<b>aio_return()</b>	409
asynchronous I/O request(s)	(POSIX). wait for .....	<b>aio_suspend()</b>	410
initiate asynchronous write	(POSIX).....	<b>aio_write()</b>	410
broken-down time into string	(POSIX). convert .....	<b>asctime_r()</b>	416
get clock resolution	(POSIX).....	<b>clock_getres()</b>	494
get current time of clock	(POSIX).....	<b>clock_gettime()</b>	495
set clock to specified time	(POSIX).....	<b>clock_settime()</b>	496
close directory	(POSIX).....	<b>closedir()</b>	497
time in seconds into string	(POSIX). convert .....	<b>ctime_r()</b>	510
open file specified by fd	(POSIX).....	<b>fdopen()</b>	587
return fd for stream	(POSIX).....	<b>fileno()</b>	591
get file status information	(POSIX).....	<b>fstat()</b>	619
clock library	(POSIX).....	<b>clockLib</b>	61
get file status information	(POSIX).....	<b>fstatfs()</b>	620
truncate file	(POSIX).....	<b>ftruncate()</b>	633
get current default path	(POSIX).....	<b>getcwd()</b>	636
time into broken-down time	(POSIX). convert calendar .....	<b>gmtime_r()</b>	641
send signal to task	(POSIX).....	<b>kill()</b>	728
asynchronous I/O requests	(POSIX). initiate list of .....	<b>lio_listio()</b>	735
time into broken-down time	(POSIX). convert calendar .....	<b>localtime_r()</b>	746
directory handling library	(POSIX).....	<b>dirLib</b>	81
specified pages into memory	(POSIX). lock .....	<b>mlock()</b>	822
used by process into memory	(POSIX). lock all pages .....	<b>mlockall()</b>	823
close message queue	(POSIX).....	<b>mq_close()</b>	838
get message queue attributes	(POSIX).....	<b>mq_getattr()</b>	839
message is available on queue	(POSIX). notify task that.....	<b>mq_notify()</b>	840
open message queue	(POSIX).....	<b>mq_open()</b>	841
message from message queue	(POSIX). receive.....	<b>mq_receive()</b>	842
send message to message queue	(POSIX).....	<b>mq_send()</b>	843
set message queue attributes	(POSIX).....	<b>mq_setattr()</b>	844
remove message queue	(POSIX).....	<b>mq_unlink()</b>	845
unlock specified pages	(POSIX).....	<b>munlock()</b>	869
all pages used by process	(POSIX). unlock .....	<b>munlockall()</b>	869
until time interval elapses	(POSIX). suspend current task .....	<b>nanosleep()</b>	906
open directory for searching	(POSIX).....	<b>opendir()</b>	946
task until delivery of signal	(POSIX). suspend .....	<b>pause()</b>	950
asynchronous I/O (AIO) library	(POSIX).....	<b>aioPxLib</b>	9
	POSIX file truncation .....	<b>ftruncate</b>	117
initialize	POSIX message queue library.....	<b>mqPxLibInit()</b>	837
	POSIX message queue show.....	<b>mqPxShow</b>	180
facility. initialize	POSIX message queue show.....	<b>mqPxShowInit()</b>	838
initialize	POSIX semaphore show facility.....	<b>semPxShowInit()</b>	1186
	POSIX semaphore show library.....	<b>semPxShow</b>	273
initialize	POSIX semaphore support.....	<b>semPxLibInit()</b>	1185
initialize	POSIX threads support .....	<b>pthreadLibInit()</b>	1007
tools.	post user event string to host .....	<b>wdbUserEvtPost()</b>	1463
message is sent. perform	post-processing after SCSI .....	<b>scsiMsgOutComplete()</b>	1150
message is rejected. perform	post-processing when outgoing .....	<b>scsiMsgOutReject()</b>	1151

	Keyword	Name	Page
of number raised to specified	power (ANSI). compute value .....	<b>pow()</b>	981
of number raised to specified	power (ANSI). compute value .....	<b>powf()</b>	982
(PowerPC, SH, x86). get	power management mode.....	<b>vxPowerModeGet()</b>	1456
(PowerPC, SH, x86). set	power management mode.....	<b>vxPowerModeSet()</b>	1456
into normalized fraction and	power of 2 (ANSI). /number .....	<b>frexp()</b>	613
multiply number by integral	power of 2 (ANSI).....	<b>ldexp()</b>	731
to critical exception vector	(PowerPC 403). /C routine.....	<b>excCrtConnect()</b>	578
to critical interrupt vector	(PowerPC 403). /C routine.....	<b>excIntCrtConnect()</b>	581
C routine to exception vector	(PowerPC). connect .....	<b>excConnect()</b>	577
asynchronous exception vector	(PowerPC, ARM). /C routine to.....	<b>excIntConnect()</b>	580
get CPU exception vector	(PowerPC, ARM). .....	<b>excVecGet()</b>	582
set CPU exception vector	(PowerPC, ARM). .....	<b>excVecSet()</b>	584
interrupt bits (MIPS,	PowerPC, ARM). /corresponding.....	<b>intDisable()</b>	687
interrupt bits (MIPS,	PowerPC, ARM). /corresponding.....	<b>intEnable()</b>	688
in reduced-power mode	(PowerPC, SH). /processor .....	<b>vxPowerDown()</b>	1455
get power management mode	(PowerPC, SH, x86). .....	<b>vxPowerModeGet()</b>	1456
set power management mode	(PowerPC, SH, x86). .....	<b>vxPowerModeSet()</b>	1456
library.	PPP authentication secrets.....	<b>pppSecretLib</b>	214
table. add secret to	PPP authentication secrets.....	<b>pppSecretAdd()</b>	993
table. delete secret from	PPP authentication secrets.....	<b>pppSecretDelete()</b>	994
table. display	PPP authentication secrets.....	<b>pppSecretShow()</b>	994
	PPP hook library.....	<b>pppHookLib</b>	212
get	PPP link statistics. ....	<b>pppstatGet()</b>	995
display	PPP link statistics. ....	<b>pppstatShow()</b>	995
get	PPP link status information.....	<b>pppInfoGet()</b>	984
display	PPP link status information.....	<b>pppInfoShow()</b>	985
delete	PPP network interface. ....	<b>pppDelete()</b>	982
initialize	PPP network interface. ....	<b>pppInit()</b>	985
register mach (also macl,	pr) (SH). /contents of system.....	<b>mach()</b>	802
answer. send	pre-formatted query and return .....	<b>resolvSend()</b>	1081
name events (WindView).	preserve extra copy of task.....	<b>wvTaskNamesPreserve()</b>	1484
(WindView). upload	preserved task name events .....	<b>wvTaskNamesUpload()</b>	1485
filtering.	prevent strict border gateway .....	<b>ripFilterDisable()</b>	1092
set	primary logging fd.....	<b>logFdSet()</b>	750
C-callable atomic test-and-set	primitive.....	<b>vxTas()</b>	1459
from task's TCB.	print complete information .....	<b>ti()</b>	1378
registers.	print contents of task's DSP.....	<b>dspTaskRegsShow()</b>	566
floating-point registers.	print contents of task's .....	<b>fppTaskRegsShow()</b>	606
directory.	print current default .....	<b>pwd()</b>	1048
error status value.	print definition of specified .....	<b>printErrno()</b>	996
device.	print information about CBIO.....	<b>cbioShow()</b>	484
cache.	print information about disk.....	<b>dcacheShow()</b>	519
object.	print information on specified .....	<b>show()</b>	1215
stack usage.	print summary of each task's.....	<b>checkStack()</b>	491
TCB.	print summary of each task's.....	<b>i()</b>	656
functions.	print synopsis of I/O utility .....	<b>ioHelp()</b>	707
routines.	print synopsis of network.....	<b>netHelp()</b>	914
routines.	print synopsis of selected .....	<b>help()</b>	651
	print VxWorks logo. ....	<b>printLogo()</b>	1001
information.	print VxWorks version .....	<b>version()</b>	1430

	Keyword	Name	Page
test whether character is	printable, including space/ .....	<b>isprint()</b>	723
test whether character is	printing, non-white-space/ .....	<b>isgraph()</b>	722
attr object (POSIX). get value of	prioceiling attr in mutex.....	<b>pthread_mutexattr_getprioceiling()</b>	1035
attributes object/ set	prioceiling attr in mutex.....	<b>pthread_mutexattr_setprioceiling()</b>	1037
(POSIX). get value of	prioceiling attribute of mutex... ..	<b>pthread_mutex_getprioceiling()</b>	1031
(POSIX). dynamically set	prioceiling attribute of mutex... ..	<b>pthread_mutex_setprioceiling()</b>	1033
tWVUpload task/ set	priority and stack size of .....	<b>wvUploadTaskConfig()</b>	1488
network events with specific	priority level. enable .....	<b>wvNetLevelAdd()</b>	1479
network events with specific	priority level. disable .....	<b>wvNetLevelRemove()</b>	1479
examine	priority of task. ....	<b>taskPriorityGet()</b>	1339
change	priority of task. ....	<b>taskPrioritySet()</b>	1339
get	priority of tMuxPollTask. ....	<b>muxTaskPriorityGet()</b>	894
reset	priority of tMuxPollTask. ....	<b>muxTaskPrioritySet()</b>	895
manager (WindView). set	priority of WindView rBuff.....	<b>wvRBuffMgrPrioritySet()</b>	1483
get maximum	priority (POSIX).....	<b>sched_get_priority_max()</b>	1131
get minimum	priority (POSIX).....	<b>sched_get_priority_min()</b>	1132
set task's	priority (POSIX).....	<b>sched_setparam()</b>	1134
create	private environment. ....	<b>envPrivateCreate()</b>	570
destroy	private environment. ....	<b>envPrivateDestroy()</b>	570
bind socket to	privileged IP port. ....	<b>bindresvport()</b>	431
open socket with	privileged port bound to it. ....	<b>rresvport()</b>	1125
	probe address for bus error.....	<b>vxMemProbe()</b>	1454
floating-point coprocessor.	probe for presence of .....	<b>fppProbe()</b>	601
library. Remote	Procedure Call (RPC) support.....	<b>rpcLib</b>	239
initialize cache library for	processor architecture.....	<b>cacheLibInit()</b>	457
mode (PowerPC, SH). place	processor in reduced-power .....	<b>vxPowerDown()</b>	1455
get	processor number. ....	<b>sysProcNumGet()</b>	1311
set	processor number. ....	<b>sysProcNumSet()</b>	1311
return contents of current	processor status register / .....	<b>cpsr()</b>	508
determine	processor time in use (ANSI).....	<b>clock()</b>	494
pass string to command	processor (Unimplemented) / .....	<b>system()</b>	1317
memory. flush	processor write buffers to.....	<b>cachePipeFlush()</b>	460
mapping library for ARM Ltd.	processors. MMU .....	<b>mmuMapLib</b>	162
cause abnormal	program termination (ANSI).....	<b>abort()</b>	403
call function at	program termination / .....	<b>atexit()</b>	421
return contents of	program counter. ....	<b>pc()</b>	950
put diagnostics into	programs (ANSI).....	<b>assert()</b>	418
change shell	prompt. ....	<b>shellPromptSet()</b>	1214
entry. display login	prompt and validate user.....	<b>loginPrompt()</b>	753
parameters.	prompt for boot line.....	<b>bootParamsPrompt()</b>	435
debugging information for TCP	protocol. display .....	<b>tcpDebugShow()</b>	1357
display all statistics for TCP	protocol.....	<b>tcpstatShow()</b>	1358
display statistics for UDP	protocol.....	<b>udpstatShow()</b>	1413
interface between BSD IP	protocol and MUX.....	<b>ipProto</b>	128
proxy Address Resolution	Protocol (ARP) client library.....	<b>proxyLib</b>	216
proxy Address Resolution	Protocol (ARP) server library.....	<b>proxyArpLib</b>	215
Address Resolution	Protocol (ARP) table / .....	<b>arpLib</b>	26
attribute object (POSIX). set	protocol attribute in mutex.....	<b>pthread_mutexattr_setprotocol()</b>	1037
library. Bootstrap	Protocol (BOOTP) client .....	<b>bootpLib</b>	33
Dynamic Host Configuration	Protocol (DHCP) run-time / .....	<b>dhcpcLib</b>	73

	Keyword	Name	Page
Dynamic Host Configuration Protocol (DHCP) server/	dhcpsLib		76
File Transfer Protocol (FTP) library.	ftpLib		115
File Transfer Protocol (FTP) server.	ftpdLib		113
object (POSIX). get value of protocol in mutex attributes ....	pthread_mutexattr_getprotocol()		1036
mount protocol library.	mountLib		178
Point-to-Point Protocol library.	pppLib		212
to receive all internet protocol packets. add routine.....	ipFilterHookAdd()		715
library. Routing Information Protocol (RIP) v1 and v2	ripLib		234
Trivial File Transfer Protocol server library.	tftpdLib		318
Point-to-Point Protocol show routines.....	pppShow		215
library. Simple Network Time Protocol (SNTP) client.....	sntpLib		291
library. Simple Network Time Protocol (SNTP) server.....	sntpsLib		292
connections for Internet protocol sockets. /all active.....	inetstatShow()		681
Trivial File Transfer Protocol (TFTP) client/	tftpdLib		319
bind NPT protocol to driver.	muxTkBind()		895
routing table. add protocol-specific route to	mRouteEntryAdd()		848
Protocol (ARP) client/ proxy Address Resolution	proxyLib		216
Protocol (ARP) server/ proxy Address Resolution	proxyArpLib		215
initialize proxy ARP.	proxyArpLibInit()		1001
create proxy ARP network.....	proxyNetCreate()		1002
show proxy ARP networks.	proxyNetShow()		1003
register proxy client.	proxyReg()		1005
unregister proxy client.	proxyUnreg()		1005
delete proxy network.	proxyNetDelete()		1002
create pseudo terminal.	ptyDevCreate()		1043
destroy pseudo terminal.	ptyDevRemove()		1044
0 and RAND_MAX/ generate pseudo-memory device driver.....	memDrv		156
initialize pseudo-random integer between	rand()		1054
pseudo-terminal driver.	ptyDrv()		1044
pseudo-terminal driver.	ptyDrv		223
display meaning of specified psr value, symbolically (ARM).	psrShow()		1006
show state of Pty Buffers.....	ptyShow()		1045
inflate code using public domain zlib functions.	inflateLib		123
bus. pulse reset signal on SCSI	scsiBusReset()		1139
test whether character is punctuation (ANSI).	ispunct()		723
create all types of DNS queries.	resolvMkQuery()		1078
send pre-formatted query and return answer.....	resolvSend()		1081
of IP address. query DNS server for host name	resolvGetHostByAddr()		1075
address of host. query DNS server for IP.....	resolvGetHostByName()		1076
response. construct query, send it, wait for.....	resolvQuery()		1080
trigger work queue task and queue. reset	trgWorkQReset()		1400
create and initialize message queue.	msgQCreate()		850
delete message queue.	msgQDelete()		851
process for message queue. /event notification	msgQEvStart()		859
process for message queue. /event notification	msgQEvStop()		860
get information about message queue.	msgQInfoGet()		861
of messages queued to message queue. get number	msgQNumMsgs()		863
receive message from message queue.	msgQReceive()		864
send message to message queue.	msgQSend()		865
show information about message queue.	msgQShow()		866

	Keyword	Name	Page
	get message	queue attributes (POSIX).....	<b>mq_getattr()</b> 839
	set message	queue attributes (POSIX).....	<b>mq_setattr()</b> 844
	delete distributed message	queue from group (VxFusion).....	<b>msgQDistGrpDelete()</b> 854
	distributed message	queue group library / .....	<b>msgQDistGrpLib</b> 180
	distributed message	queue group show routines / .....	<b>msgQDistGrpShow</b> 181
	message	queue library. ....	<b>msgQLib</b> 183
	initialize POSIX message	queue library. ....	<b>mqPxLibInit()</b> 837
	message	queue library (POSIX).....	<b>mqPxLib</b> 179
	distributed objects message	queue library (VxFusion).....	<b>msgQDistLib</b> 181
	shared memory message	queue library (VxMP).....	<b>msgQSmLib</b> 185
	close message	queue (POSIX).....	<b>mq_close()</b> 838
	that message is available on	queue (POSIX). notify task.....	<b>mq_notify()</b> 840
	open message	queue (POSIX).....	<b>mq_open()</b> 841
	receive message from message	queue (POSIX).....	<b>mq_receive()</b> 842
	send message to message	queue (POSIX).....	<b>mq_send()</b> 843
	remove message	queue (POSIX).....	<b>mq_unlink()</b> 845
	POSIX message	queue show.....	<b>mqPxShow</b> 180
	initialize POSIX message	queue show facility.....	<b>mqPxShowInit()</b> 838
	initialize message	queue show facility.....	<b>msgQShowInit()</b> 867
	initialize distributed message	queue show package (VxFusion).....	<b>msgQDistShowInit()</b> 859
	message	queue show routines.....	<b>msgQShow</b> 185
	distributed message	queue show routines / .....	<b>msgQDistShow</b> 182
	reset trigger work	queue task and queue.....	<b>trgWorkQReset()</b> 1400
	add distributed message	queue to group (VxFusion).....	<b>msgQDistGrpAdd()</b> 853
	create distributed message	queue (VxFusion).....	<b>msgQDistCreate()</b> 851
	in distributed message	queue (VxFusion). /of messages.....	<b>msgQDistNumMsgs()</b> 855
	from distributed message	queue (VxFusion). /message.....	<b>msgQDistReceive()</b> 856
	message to distributed message	queue (VxFusion). send.....	<b>msgQDistSend()</b> 857
	shared memory message	queue (VxMP). /and initialize .....	<b>msgQSmCreate()</b> 868
	initialize	queued signal facilities.....	<b>sigqueueInit()</b> 1223
	send	queued signal to task .....	<b>sigqueue()</b> 1222
	get number of messages	queued to message queue.....	<b>msgQNumMsgs()</b> 863
	events support for message	queues. VxWorks.....	<b>msgQEvLib</b> 183
	BSP serial devices to	quiescent state. initialize .....	<b>sysSerialHwInit()</b> 1315
	reset all SIO devices to	quiet state. ....	<b>sysSerialReset()</b> 1316
	compute	quotient and remainder (ANSI).....	<b>div()</b> 557
	division (ANSI). compute	quotient and remainder of .....	<b>ldiv()</b> 732
	(reentrant). compute	quotient and remainder.....	<b>div_r()</b> 558
	(reentrant). compute	quotient and remainder.....	<b>ldiv_r()</b> 732
	return contents of register	r0 (also r1 - r14, r1-r15 for / .....	<b>r0()</b> 1050
	/of contents of register r0 (also	r1 - r14, r1-r15 for SH) (ARM, / .....	<b>r0()</b> 1050
	initialize	R10000 cache library.....	<b>cacheR10kLibInit()</b> 463
	library. MIPS	R10000 cache management .....	<b>cacheR10kLib</b> 49
	/of register r0 (also r1 -	r14, r1-r15 for SH) (ARM, SH).....	<b>r0()</b> 1050
	/of register r0 (also r1 - r14,	r1-r15 for SH) (ARM, SH).....	<b>r0()</b> 1050
	initialize	R3000 cache library.....	<b>cacheR3kLibInit()</b> 460
	library. MIPS	R3000 cache management .....	<b>cacheR3kLib</b> 47
	initialize	R33000 cache library.....	<b>cacheR33kLibInit()</b> 464
	library. MIPS	R33000 cache management .....	<b>cacheR33kLib</b> 50
	initialize	R333x0 cache library.....	<b>cacheR333x0LibInit()</b> 465

	Keyword	Name	Page
library. MIPS	R333x0 cache management.....	<b>cacheR333x0Lib</b>	50
initialize	R4000 cache library.....	<b>cacheR4kLibInit()</b>	461
library. MIPS	R4000 cache management.....	<b>cacheR4kLib</b>	47
initialize	R5000 cache library.....	<b>cacheR5kLibInit()</b>	461
library. MIPS	R5000 cache management.....	<b>cacheR5kLib</b>	48
initialize	R7000 cache library.....	<b>cacheR7kLibInit()</b>	462
library. MIPS	R7000 cache management.....	<b>cacheR7kLib</b>	48
get contents of non-volatile	RAM.....	<b>sysNvRamGet()</b>	1309
write to non-volatile	RAM.....	<b>sysNvRamSet()</b>	1310
	RAM Disk Cached Block Driver.....	<b>ramDiskCbio</b>	225
create	RAM disk device.....	<b>ramDevCreate()</b>	1051
initialize	RAM Disk device.....	<b>ramDiskDevCreate()</b>	1052
	RAM disk driver.....	<b>ramDrv</b>	225
(optional). prepare	RAM disk driver for use.....	<b>ramDrv()</b>	1053
/integer between 0 and	RAND_MAX (ANSI).....	<b>rand()</b>	1054
value of seed used to generate	random numbers (ANSI). reset.....	<b>srand()</b>	1262
disable MTRR (Memory Type	Range Register).....	<b>pentiumMtrrDisable()</b>	956
enable MTRR (Memory Type	Range Register).....	<b>pentiumMtrrEnable()</b>	956
library.	raw block device file system.....	<b>rawFsLib</b>	226
modify mode of	raw device volume.....	<b>rawFsModeChange()</b>	1055
disable	raw device volume.....	<b>rawFsVolUnmount()</b>	1056
associate block device with	raw volume functions.....	<b>rawFsDevInit()</b>	1054
prepare to use	raw volume library.....	<b>rawFsInit()</b>	1055
status. notify	<b>rawFsLib</b> of change in ready.....	<b>rawFsReadyChange()</b>	1056
dynamic ring buffer	(rBuff) library.....	<b>rBuffLib</b>	230
set priority of WindView	rBuff manager (WindView).....	<b>wvRBuffMgrPrioritySet()</b>	1483
initialize	RC32364 cache library.....	<b>cacheR32kLibInit()</b>	463
library. MIPS	RC32364 cache management.....	<b>cacheR32kLib</b>	49
ifRcvAddressTable. populate	rcvAddr fields for.....	<b>rcvEtherAddrGet()</b>	1058
given address. get	rcvAddress table entries for.....	<b>m2IfRcvAddrEntryGet()</b>	779
modify entries of	rcvAddressTable.....	<b>m2IfRcvAddrEntrySet()</b>	780
test that remote host is	reachable.....	<b>ping()</b>	977
from ASCII string (ANSI).	read and convert characters.....	<b>sscanf()</b>	1263
from standard input stream/	read and convert characters.....	<b>scanf()</b>	1130
from stream (ANSI).	read and convert characters.....	<b>fscanf()</b>	614
	read buffer.....	<b>fioRead()</b>	596
device.	read bytes from file or.....	<b>read()</b>	1064
tape device.	read bytes or blocks from SCSI.....	<b>scsiRdTape()</b>	1154
input stream (ANSI).	read characters from standard.....	<b>gets()</b>	638
from requested NFS device.	read configuration information.....	<b>nfsDevInfoGet()</b>	934
register (MIPS).	read contents of cause.....	<b>intCRGet()</b>	686
register (MIPS).	read contents of status.....	<b>intSRGet()</b>	694
	read data into array (ANSI).....	<b>fread()</b>	611
table.	read FDISK-style partition.....	<b>usrFdiskPartRead()</b>	1422
do task-level	read for tty device.....	<b>tyRead()</b>	1411
	read line with line-editing.....	<b>ledRead()</b>	734
integer) from stream.	read next word (32-bit).....	<b>getw()</b>	640
(POSIX).	read one entry from directory.....	<b>readdir()</b>	1064
initiate asynchronous	read (POSIX).....	<b>aio_read()</b>	408
command to SCSI device and	read results. /REQUEST_SENSE.....	<b>scsiReqSense()</b>	1156

Keyword	Name	Page
block device.	read sector(s) from SCSI .....	scsiRdSecs() 1154
characters from stream/	read specified number of.....	fgets() 591
	read string from file. ....	fiORdString() 595
to SCSI device. issue	READ_BLOCK_LIMITS command ...	scsiSeqReadBlockLimits() 1159
device. issue	READ_CAPACITY command to SCSI .....	scsiReadCapacity() 1155
ISO 9660 CD-ROM	read-only file system library. ....	cdromFsLib 59
set file	read/write pointer.....	lseek() 760
notify <b>rawFsLib</b> of change in	ready status. ....	rawFsReadyChange() 1056
notify <b>rt11Fs</b> of change in	ready status. ....	rt11FsReadyChange() 1128
notify <b>tapeFsLib</b> of change in	ready status. ....	tapeFsReadyChange() 1321
determine	ready status of CBIO device. ....	cbioRdyChgdGet() 482
force change in	ready status of CBIO device. ....	cbioRdyChgdSet() 483
check if task is	ready to run.....	taskIsReady() 1335
(ANSI).	reallocate block of memory .....	realloc() 1065
from shared memory system/	reallocate block of memory .....	smMemRealloc() 1235
specified partition.	reallocate block of memory in .....	memPartRealloc() 817
wait for	real-time signals.....	sigwaitinfol() 1227
add routine to be called at	reboot. ....	rebootHookAdd() 1067
	reboot support library.....	rebootLib 231
packets. add routine to	receive all internet protocol .....	ipFilterHookAdd() 715
ID information (use <b>unld()</b> to	reclaim space). delete module .....	moduleDelete() 830
Ethernet multicast address	record. delete.....	etherMultiDel() 574
SH). place processor in	reduced-power mode (PowerPC, .....	vxPowerDown() 1455
	re-enable disk cache. ....	dcacheDevEnable() 516
compute quotient and remainder	(reentrant).....	div_r() 558
compute quotient and remainder	(reentrant).....	ldiv_r() 732
break down string into tokens	(reentrant) (POSIX).....	strtok_r() 1283
assign routine to access	reference clock. ....	sntpsClockSet() 1250
get content of Debug	Register 0 to 7 (x86). ....	vxDrGet() 1450
set value to Debug	Register 0 to 7 (x86). ....	vxDrSet() 1451
get content of Control	Register 0 (x86).....	vxCr0Get() 1449
set value to Control	Register 0 (x86).....	vxCr0Set() 1450
get content of Control	Register 2 (x86).....	vxCr2Get() 1447
set value to Control	Register 2 (x86).....	vxCr2Set() 1447
get content of Control	Register 3 (x86).....	vxCr3Get() 1448
set value to Control	Register 3 (x86).....	vxCr3Set() 1448
get content of Control	Register 4 (x86).....	vxCr4Get() 1448
set value to Control	Register 4 (x86).....	vxCr4Set() 1449
set task status	register (68K, MIPS, x86). ....	taskSRSet() 1347
return contents of status	register (68K, SH). ....	sr() 1262
get contents of CR4	register.....	pentiumCr4Get() 952
sets specified value to CR4	register.....	pentiumCr4Set() 952
specified MSR (Model Specific	Register). get contents of .....	pentiumMsrGet() 954
all MSRs (Model Specific	Register). initialize.....	pentiumMsrInit() 954
show all MSR (Model Specific	Register).....	pentiumMsrShow() 955
MTRR (Memory Type Range	Register). disable .....	pentiumMtrrDisable() 956
MTRR (Memory Type Range	Register). enable .....	pentiumMtrrEnable() 956
(68K). return contents of	register a0 (also a1 - a7) .....	a0() 403
of current processor status	register (ARM). /contents.....	cpsr() 508
(68K). return contents of	register d0 (also d1 - d7) .....	d0() 512

	Keyword	Name	Page
	return contents of register edi (also esi - eax)/	<b>edi()</b>	568
	return contents of system register mach (also macl, pr)/	<b>mach()</b>	802
	initialize default task status register (MIPS).	<b>taskSRInit()</b>	1346
	read contents of cause register (MIPS).	<b>intCRGet()</b>	686
	write contents of cause register (MIPS).	<b>intCRSet()</b>	687
	read contents of status register (MIPS).	<b>intSRGet()</b>	694
	update contents of status register (MIPS).	<b>intSRSet()</b>	694
	register proxy client.	<b>proxyReg()</b>	1005
r1-r15 for/	return contents of register r0 (also r1 - r14,	<b>r0()</b>	1050
(WindView).	register timestamp timer.	<b>wvTmrRegister()</b>	1485
get content of EFLAGS	register (x86).	<b>vxEflagsGet()</b>	1451
set value to EFLAGS	register (x86).	<b>vxEflagsSet()</b>	1452
of Global Descriptor Table	Register (x86). get content.	<b>vxGdtrGet()</b>	1452
of Interrupt Descriptor Table	Register (x86). get content.	<b>vxIdtrGet()</b>	1453
of Local Descriptor Table	Register (x86). get content.	<b>vxLdtrGet()</b>	1453
get content of TASK	register (x86).	<b>vxTssGet()</b>	1460
set value to TASK	register (x86).	<b>vxTssSet()</b>	1460
return contents of status	register (x86/SimNT).	<b>eflags()</b>	568
routing system. remove	registered handler from	<b>routeStorageUnbind()</b>	1122
configuration of devices	registered with MUX. display	<b>muxShow()</b>	893
set task's	registers.	<b>taskRegsSet()</b>	1340
display contents of task's	registers.	<b>taskRegsShow()</b>	1341
print contents of task's DSP	registers.	<b>dspTaskRegsShow()</b>	566
of task's floating-point	registers. print contents.	<b>fppTaskRegsShow()</b>	606
modify	registers.	<b>mRegs()</b>	845
(Machine Check Architecture)	registers. show MCA	<b>pentiumMcaShow()</b>	953
specified MSR (Model Specific	Registers). set value to	<b>pentiumMsrSet()</b>	955
get floating-point	registers from task TCB.	<b>fppTaskRegsGet()</b>	605
get task's	registers from TCB.	<b>taskRegsGet()</b>	1340
set floating-point	registers of task.	<b>fppTaskRegsSet()</b>	605
DHCP	relay agent library.	<b>dhcprLib</b>	75
	relinquish CPU (POSIX).	<b>sched_yield()</b>	1136
	relinquish specified lease.	<b>dhcpcRelease()</b>	534
	reload object module.	<b>reld()</b>	1069
/time before expiration and	reload value (POSIX).	<b>timer_gettime()</b>	1384
compute quotient and	remainder (ANSI).	<b>div()</b>	557
compute quotient and	remainder of division (ANSI).	<b>ldiv()</b>	732
compute	remainder of x/y (ANSI).	<b>fmod()</b>	598
compute	remainder of x/y (ANSI).	<b>fmodf()</b>	598
compute quotient and	remainder (reentrant).	<b>div_r()</b>	558
compute quotient and	remainder (reentrant).	<b>ldiv_r()</b>	732
hashing function using	remainder technique.	<b>hashFuncModulo()</b>	644
expiration and reload/ get	remaining time before	<b>timer_gettime()</b>	1384
	remote access to target shell.	<b>remShellLib</b>	232
	remote command library.	<b>remLib</b>	231
create	remote file device.	<b>netDevCreate()</b>	911
buffer size. create	remote file device with fixed	<b>netDevCreate2()</b>	912
install network	remote file driver.	<b>netDrv()</b>	913
network	remote file I/O driver.	<b>netDrv</b>	190
log in to	remote FTP server.	<b>ftpLogin()</b>	627

	Keyword	Name	Page
	log in to remote host.....	<b>rlogin()</b>	1105
exported file systems of	remote host. display.....	<b>nfsExportShow()</b>	939
test that	remote host is reachable.....	<b>ping()</b>	977
display current	remote identity.....	<b>whoami()</b>	1468
VxWorks	remote login daemon.....	<b>rlogind()</b>	1105
initialize	remote login facility.....	<b>rlogInit()</b>	1106
	remote login library.....	<b>rlogLib</b>	236
execute shell command on	remote machine.....	<b>rcmd()</b>	1057
support library.	Remote Procedure Call (RPC).....	<b>rpcLib</b>	239
retrieve current time from	remote source.....	<b>sntpTimeGet()</b>	1249
get file from	remote system.....	<b>tftpGet()</b>	1372
put file to	remote system.....	<b>tftpPut()</b>	1374
send TFTP message to	remote system.....	<b>tftpSend()</b>	1375
set	remote user name and password.....	<b>remCurIdSet()</b>	1070
set	remote user name and password.....	<b>iam()</b>	657
events.	remove address filter for.....	<b>wvNetAddressFilterClear()</b>	1475
	remove ARP table entry.....	<b>arpDelete()</b>	413
from RIP interface.	remove authentication hook.....	<b>ripAuthHookDelete()</b>	1087
parameters handler.	remove configuration.....	<b>dhcpcEventHookDelete()</b>	526
	remove directory.....	<b>rmdir()</b>	1107
name database (VxFusion).	remove entry from distributed.....	<b>distNameRemove()</b>	555
insert or	remove entry in ifTable.....	<b>m2IfTableUpdate()</b>	783
	remove file.....	<b>rm()</b>	1107
	remove file (ANSI).....	<b>remove()</b>	1071
list of exported file/	remove file system from.....	<b>nfsUnexport()</b>	942
table.	remove hash node from hash.....	<b>hashTblRemove()</b>	650
	remove I/O driver.....	<b>iosDrvRemove()</b>	710
	remove message queue (POSIX).....	<b>mq_unlink()</b>	845
(POSIX).	remove named semaphore.....	<b>sem_unlink()</b>	1194
memory objects name database/	remove object from shared.....	<b>smNameRemove()</b>	1240
events.	remove port number filter for.....	<b>wvNetPortFilterClear()</b>	1480
timer (POSIX).	remove previously created.....	<b>timer_delete()</b>	1383
routing system.	remove registered handler from.....	<b>routeStorageUnbind()</b>	1122
table.	remove route from routing.....	<b>routeEntryDel()</b>	1117
	remove route hook.....	<b>ripRouteHookDelete()</b>	1102
table.	remove symbol from symbol.....	<b>symRemove()</b>	1294
RIP interface.	remove table bypass hook from.....	<b>ripLeakHookDelete()</b>	1097
task.	remove task variable from.....	<b>taskVarDelete()</b>	1354
	remove tty device descriptor.....	<b>tyDevRemove()</b>	1408
interface.	remove update filter from RIP.....	<b>ripSendHookDelete()</b>	1104
request message and retrieve	reply. send BOOTP.....	<b>bootpMsgGet()</b>	436
send FTP command and get	reply.....	<b>ftpCommand()</b>	621
get FTP command	reply.....	<b>ftpReplyGet()</b>	628
get FTP command	reply.....	<b>ftpReplyGetEnhanced()</b>	629
command and get complete RFC	reply code. send FTP.....	<b>ftpCommandEnhanced()</b>	622
wait for asynchronous I/O	request(s) (POSIX).....	<b>aio_suspend()</b>	410
device and read/ issue	REQUEST_SENSE command to SCSI.....	<b>scsiReqSense()</b>	1156
disable task	rescheduling.....	<b>taskLock()</b>	1336
enable task	rescheduling.....	<b>taskUnlock()</b>	1351
device. issue	RESERVE command to SCSI.....	<b>scsiReserve()</b>	1157

	Keyword	Name	Page
device. issue	RESERVE UNIT command to SCSI.....	<b>scsiReserveUnit()</b>	1157
state.	reset all SIO devices to quiet.....	<b>sysSerialReset()</b>	1316
	reset both PMC0 and PMC1.....	<b>pentiumP5PmcReset()</b>	959
	reset both PMC0 and PMC1.....	<b>pentiumP6PmcReset()</b>	965
	reset both PMC0 and PMC1.....	<b>pentiumPmcReset()</b>	969
handle controller-bus	reset event.....	<b>scsiMgrBusReset()</b>	1146
transfer control to boot/	reset network devices and.....	<b>reboot()</b>	1066
	reset PMC0.....	<b>pentiumP5PmcReset0()</b>	960
	reset PMC0.....	<b>pentiumP6PmcReset0()</b>	965
	reset PMC0.....	<b>pentiumPmcReset0()</b>	969
	reset PMC1.....	<b>pentiumP5PmcReset1()</b>	960
	reset PMC1.....	<b>pentiumP6PmcReset1()</b>	965
	reset PMC1.....	<b>pentiumPmcReset1()</b>	969
directory (POSIX).	reset position to start of.....	<b>rewinddir()</b>	1082
tMuxPollTask.	reset priority of.....	<b>muxTaskPrioritySet()</b>	895
pulse	reset signal on SCSI bus.....	<b>scsiBusReset()</b>	1139
and queue.	reset trigger work queue task.....	<b>trgWorkQReset()</b>	1400
	reset TSC (Timestamp Counter).....	<b>pentiumTscReset()</b>	974
generate random numbers/	reset value of seed used to.....	<b>srand()</b>	1262
set clock	resolution.....	<b>clock_setres()</b>	495
replace default address	resolution function.....	<b>muxAddrResFuncAdd()</b>	871
delete address	resolution function.....	<b>muxAddrResFuncDel()</b>	872
ifType/protocol. get address	resolution function for.....	<b>muxAddrResFuncGet()</b>	873
get clock	resolution (POSIX).....	<b>clock_getres()</b>	494
client library. proxy Address	Resolution Protocol (ARP).....	<b>proxyLib</b>	216
server library. proxy Address	Resolution Protocol (ARP).....	<b>proxyArpLib</b>	215
table manipulation/ Address	Resolution Protocol (ARP).....	<b>arpLib</b>	26
initialize	resolver library.....	<b>resolvInit()</b>	1077
get parameters which control	resolver library.....	<b>resolvParamsGet()</b>	1078
set parameters which control	resolver library.....	<b>resolvParamsSet()</b>	1079
DNS	resolver library.....	<b>resolvLib</b>	232
	restart task.....	<b>taskRestart()</b>	1341
delete task without	restriction.....	<b>taskDeleteForce()</b>	1328
/semaphore without	restrictions.....	<b>semMGiveForce()</b>	1185
return BSP version and	revision number.....	<b>sysBspRev()</b>	1299
return kernel	revision string.....	<b>kernelVersion()</b>	727
issue	REWIND command to SCSI device.....	<b>scsiRewind()</b>	1158
FTP command and get complete	RFC reply code. send.....	<b>ftpCommandEnhanced()</b>	622
get characters from	ring buffer.....	<b>rngBufGet()</b>	1108
put bytes into	ring buffer.....	<b>rngBufPut()</b>	1108
create empty	ring buffer.....	<b>rngCreate()</b>	1109
delete	ring buffer.....	<b>rngDelete()</b>	1109
number of free bytes in	ring buffer. determine.....	<b>rngFreeBytes()</b>	1110
determine number of bytes in	ring buffer.....	<b>rngNBytes()</b>	1112
make	ring buffer empty.....	<b>rngFlush()</b>	1110
test if	ring buffer is empty.....	<b>rngIsEmpty()</b>	1111
room). test if	ring buffer is full (no more.....	<b>rngIsFull()</b>	1111
dynamic	ring buffer (rBuff) library.....	<b>rBuffLib</b>	230
library.	ring buffer subroutine.....	<b>rngLib</b>	237
ring/ put byte ahead in	ring buffer without moving.....	<b>rngPutAhead()</b>	1113

	Keyword	Name	Page
	advance	ring pointer by n bytes. ....	rngMoveAhead() 1112
in ring buffer without moving		ring pointers. put byte ahead .....	rngPutAhead() 1113
interface table maintained by		RIP. display internal .....	ripIfShow() 1096
static and non-RIP routes into		RIP. add hook to install.....	ripRouteHookAdd() 1099
routing table maintained by		RIP. display internal .....	ripRouteShow() 1102
add hook to bypass		RIP and kernel routing tables.....	ripLeakHookAdd() 1096
add new		RIP authentication key.....	ripAuthKeyAdd() 1088
delete existing		RIP authentication key.....	ripAuthKeyDelete() 1088
find		RIP authentication key.....	ripAuthKeyFind() 1089
find		RIP authentication key.....	ripAuthKeyFindFirst() 1089
interface changes. alter		RIP configuration after .....	ripIfReset() 1095
add interface to		RIP exclusion list. ....	ripIfExcludeListAdd() 1093
delete interface from		RIP exclusion list. ....	ripIfExcludeListDelete() 1094
VxWorks interface routines to		RIP for SNMP Agent.....	m2RipLib 147
add authentication hook to		RIP interface. ....	ripAuthHookAdd() 1085
authentication hook from		RIP interface. remove.....	ripAuthHookDelete() 1087
remove table bypass hook from		RIP interface. ....	ripLeakHookDelete() 1097
add update filter to		RIP interface. ....	ripSendHookAdd() 1103
remove update filter from		RIP interface. ....	ripSendHookDelete() 1104
show		RIP interface exclusion list.....	ripIfExcludeListShow() 1094
delete		RIP MIB support.....	m2RipDelete() 792
initialize		RIP MIB support.....	m2RipInit() 795
terminate all		RIP processing. ....	ripShutdown() 1104
initialize		RIP routing library.....	ripLibInit() 1097
Routing Information Protocol		(RIP) v1 and v2 library.....	ripLib 234
MD5 authentication of outgoing		RIP-2 message. start .....	ripAuthKeyOut1MD5() 1090
authenticate incoming		RIP-2 message using MD5. ....	ripAuthKeyInMD5() 1090
authenticate outgoing		RIP-2 message using MD5. ....	ripAuthKeyOut2MD5() 1091
get MIB-II		RIP-group global counters.....	m2RipGlobalCountersGet() 793
get MIB-II		RIP-group interface entry.....	m2RipIfConfEntryGet() 793
set MIB-II		RIP-group interface entry.....	m2RipIfConfEntrySet() 794
get MIB-II		RIP-group interface entry.....	m2RipIfStatEntryGet() 794
generic		ROM initialization.....	romStart() 1113
		ROM initialization module.....	bootInit 30
transfer control to		ROM monitor. ....	sysToMonitor() 1317
boot		ROM subroutine library. ....	bootLib 31
and transfer control to boot		ROMs. reset network devices .....	reboot() 1066
configuration module for boot		ROMs. system .....	bootConfig 29
ring buffer is full (no more		room). test if .....	rngIsFull() 1111
compute cube		root.....	cbrt() 485
compute cube		root.....	cbrtf() 486
compute non-negative square		root (ANSI).....	sqrt() 1261
compute non-negative square		root (ANSI).....	sqrtrf() 1261
		root task. ....	usrRoot() 1425
integer.		round number to nearest.....	round() 1114
integer.		round number to nearest.....	roundf() 1114
integer.		round number to nearest.....	iround() 718
integer.		round number to nearest.....	iroundf() 719
enable		round-robin selection.....	kernelTimeSlice() 727
add		route. ....	routeAdd() 1115

	Keyword	Name	Page
	delete route.....	<b>routeDelete()</b>	1116
find matching	route for destination.....	<b>routeEntryLookup()</b>	1118
remove	route from routing table.....	<b>routeEntryDel()</b>	1117
	delete route from routing table.....	<b>mRouteDelete()</b>	847
	delete route from routing table.....	<b>mRouteEntryDelete()</b>	849
remove	route hook.....	<b>ripRouteHookDelete()</b>	1102
insert	route in routing table.....	<b>routeEntryAdd()</b>	1116
multiple matching entries.	route interface library for.....	<b>routeEntryLib</b>	238
network	route manipulation library.....	<b>routeLib</b>	238
socket address pointers	route message. extract.....	<b>ripAdrsXtract()</b>	1083
network. add	route to destination that is.....	<b>routeNetAdd()</b>	1120
add protocol-specific	route to routing table.....	<b>mRouteEntryAdd()</b>	848
new or removed interfaces for	router discovery. check for.....	<b>rdiscIfReset()</b>	1062
initialize	router discovery.....	<b>rdiscLibInit()</b>	1063
function. implement ICMP	router discovery control.....	<b>rdCtl()</b>	1059
implement ICMP	router discovery function.....	<b>rdisc()</b>	1061
initialize ICMP	router discovery function.....	<b>rdiscInit()</b>	1062
library. ICMP	router discovery server.....	<b>rdiscLib</b>	230
interface. delete all	routes associated with network.....	<b>ifAllRoutesDelete()</b>	660
interface. delete	routes associated with network.....	<b>ifRouteDelete()</b>	670
to install static and non-RIP	routes into RIP. add hook.....	<b>ripRouteHookAdd()</b>	1099
display all IP	routes (summary information).....	<b>routeShow()</b>	1120
add multiple	routes to same destination.....	<b>mRouteAdd()</b>	846
display all IP	routes (verbose information).....	<b>mRouteShow()</b>	849
(RIP) v1 and v2 library.	Routing Information Protocol.....	<b>ripLib</b>	234
message routines for	routing interface library.....	<b>routeMessageLib</b>	238
initialize RIP	routing library.....	<b>ripLibInit()</b>	1097
display	routing statistics.....	<b>routestatShow()</b>	1122
remove registered handler from	routing system.....	<b>routeStorageUnbind()</b>	1122
insert route in	routing table.....	<b>routeEntryAdd()</b>	1116
remove route from	routing table.....	<b>routeEntryDel()</b>	1117
change entry in	routing table.....	<b>routeModify()</b>	1119
traverse IP	routing table.....	<b>routeTableWalk()</b>	1123
delete route from	routing table.....	<b>mRouteDelete()</b>	847
add protocol-specific route to	routing table.....	<b>mRouteEntryAdd()</b>	848
delete route from	routing table.....	<b>mRouteEntryDelete()</b>	849
get MIB-2	routing table entry.....	<b>m2IpRouteTblEntryGet()</b>	791
set MIB-II	routing table entry.....	<b>m2IpRouteTblEntrySet()</b>	791
RIP. display internal	routing table maintained by.....	<b>ripRouteShow()</b>	1102
hook to bypass RIP and kernel	routing tables. add.....	<b>ripLeakHookAdd()</b>	1096
initialize	RPC package.....	<b>rpclnit()</b>	1124
initialize task's access to	RPC package.....	<b>rpcTaskInit()</b>	1124
Remote Procedure Call	(RPC) support library.....	<b>rpcLib</b>	239
Digital WD33C93 only). assert	RST line on SCSI bus (Western.....	<b>sysScsiBusReset()</b>	1312
system library.	RT-11 media-compatible file.....	<b>rt11FsLib</b>	239
initialize	rt11Fs device descriptor.....	<b>rt11FsDevInit()</b>	1126
initialize device and create	rt11Fs file system.....	<b>rt11FsMkfs()</b>	1127
set	rt11Fs file system date.....	<b>rt11FsDateSet()</b>	1125
prepare to use	rt11Fs library.....	<b>rt11FsInit()</b>	1127
status. notify	rt11Fs of change in ready.....	<b>rt11FsReadyChange()</b>	1128

	Keyword	Name	Page
	modify mode of	rt11Fs volume.....	rt11FsModeChange() 1128
	make calling task	safe from deletion.....	taskSafe() 1342
	get MIB-II interface-group	scalar variables.....	m2IfGroupInfoGet() 778
	get MIB-II IP-group	scalar variables.....	m2IpGroupInfoGet() 789
	get MIB-II TCP-group	scalar variables.....	m2TcpGroupInfoGet() 799
	get MIB-II UDP-group	scalar variables.....	m2UdpGroupInfoGet() 800
	strings. iterative	scaling hashing function for.....	hashFuncIterScale() 643
	floating-point formatting and	scanning library.....	floatLib 109
	thread/ dynamically set	schedparam attribute for.....	pthread_setschedparam() 1040
	thread (POSIX). get value of	schedparam attribute from.....	pthread_getschedparam() 1027
	attributes/ get value of	schedparam attribute in thread...	pthread_attr_getschedparam() 1009
	attributes object/ set	schedparam attribute in thread...	pthread_attr_setschedparam() 1015
	thread attributes object/ get	schedpolicy attribute from.....	pthread_attr_getschedpolicy() 1010
	thread attributes object/ set	schedpolicy attribute in.....	pthread_attr_setschedpolicy() 1016
		scheduling library (POSIX).....	schedPxBLib 245
	specified task (POSIX). get	scheduling parameters for.....	sched_getparam() 1132
	set scheduling policy and	scheduling parameters (POSIX).....	sched_setscheduler() 1135
	scheduling parameters/ set	scheduling policy and.....	sched_setscheduler() 1135
	get current	scheduling policy (POSIX).....	sched_getscheduler() 1133
	(POSIX). set contention	scope for thread attributes.....	pthread_attr_setscope() 1017
	(POSIX). get contention	scope from thread attributes.....	pthread_attr_getscope() 1011
	shell to stop processing	script. signal.....	shellScriptAbort() 1215
	define logical partition on	SCSI block device.....	scsiBlkDevCreate() 1137
	read sector(s) from	SCSI block device.....	scsiRdSecs() 1154
	write sector(s) to	SCSI block device.....	scsiWrtSecs() 1168
	pulse reset signal on	SCSI bus.....	scsiBusReset() 1139
	WD33C93/ assert RST line on	SCSI bus (Western Digital.....	sysScsiBusReset() 1312
	system	SCSI configuration.....	sysScsiConfig() 1313
	all devices connected to	SCSI controller. configure.....	scsiAutoConfig() 1137
	physical devices attached to	SCSI controller. list.....	scsiShow() 1160
	notify SCSI manager of	SCSI (controller) event.....	scsiMgrEventNotify() 1147
	send event to	SCSI controller state machine.....	scsiMgrCtrlEvent() 1146
	issue ERASE command to	SCSI device.....	scsiErase() 1142
	FORMAT_UNIT command to	SCSI device. issue.....	scsiFormatUnit() 1142
	issue INQUIRY command to	SCSI device.....	scsiInquiry() 1144
	LOAD/UNLOAD command to	SCSI device. issue.....	scsiLoadUnit() 1145
	MODE_SELECT command to	SCSI device. issue.....	scsiModeSelect() 1149
	issue MODE_SENSE command to	SCSI device.....	scsiModeSense() 1149
	READ_CAPACITY command to	SCSI device. issue.....	scsiReadCapacity() 1155
	issue RELEASE command to	SCSI device.....	scsiRelease() 1155
	RELEASE UNIT command to	SCSI device. issue.....	scsiReleaseUnit() 1156
	issue RESERVE command to	SCSI device.....	scsiReserve() 1157
	RESERVE UNIT command to	SCSI device. issue.....	scsiReserveUnit() 1157
	issue REWIND command to	SCSI device.....	scsiRewind() 1158
	command to	SCSI/ /READ_BLOCK_LIMITS.....	scsiSeqReadBlockLimits() 1159
	tape on specified physical	SCSI device. move.....	scsiSpace() 1161
	START_STOP_UNIT command to	SCSI device. issue.....	scsiStartStopUnit() 1162
	TEST_UNIT_READY command to	SCSI device. issue.....	scsiTestUnitRdy() 1166
	REQUEST_SENSE command to	SCSI device and read results. issue.....	scsiReqSense() 1156
		SCSI initialization.....	usrScsi 342

	Keyword	Name	Page
Computer System Interface	(SCSI) library. Small.....	<b>scsiLib</b>	258
for all devices (SCSI-2).	SCSI library common commands.....	<b>scsiCommonLib</b>	256
devices (SCSI-2).	SCSI library for direct access.....	<b>scsiDirectLib</b>	257
/Computer System Interface	(SCSI) library (SCSI-1).....	<b>scsi1Lib</b>	246
/Computer System Interface	(SCSI) library (SCSI-2).....	<b>scsi2Lib</b>	249
initialize fields in	SCSI logical partition.....	<b>scsiBlkDevInit()</b>	1138
show status information for	SCSI manager.....	<b>scsiMgrShow()</b>	1147
	SCSI manager library (SCSI-2).....	<b>scsiMgrLib</b>	259
(controller) event. notify	SCSI manager of SCSI.....	<b>scsiMgrEventNotify()</b>	1147
perform post-processing after	SCSI message is sent.....	<b>scsiMsgOutComplete()</b>	1150
target. handle complete	SCSI message received from.....	<b>scsiMsgInComplete()</b>	1150
configure	SCSI peripherals.....	<b>usrScsiConfig()</b>	1425
structure. create	SCSI physical device.....	<b>scsiPhysDevCreate()</b>	1151
structure. delete	SCSI physical-device.....	<b>scsiPhysDevDelete()</b>	1152
initialize on-board	SCSI port.....	<b>sysScsiInit()</b>	1314
library (SCSI-2).	SCSI sequential access device.....	<b>scsiSeqLib</b>	260
create	SCSI sequential device.....	<b>scsiSeqDevCreate()</b>	1158
write file marks to	SCSI sequential device.....	<b>scsiWrtFileMarks()</b>	1167
read bytes or blocks from	SCSI tape device.....	<b>scsiRdTape()</b>	1154
MODE_SELECT command to	SCSI tape device. issue.....	<b>scsiTapeModeSelect()</b>	1163
issue MODE_SENSE command to	SCSI tape device.....	<b>scsiTapeModeSense()</b>	1163
write data to	SCSI tape device.....	<b>scsiWrtTape()</b>	1168
display options for specified	SCSI target.....	<b>scsiTargetOptionsShow()</b>	1165
get options for one or all	SCSI targets.....	<b>scsiTargetOptionsGet()</b>	1164
set options for one or all	SCSI targets.....	<b>scsiTargetOptionsSet()</b>	1164
caches is disabled. inform	SCSI that hardware snooping of.....	<b>scsiCacheSnoopDisable()</b>	1140
caches is enabled. inform	SCSI that hardware snooping of.....	<b>scsiCacheSnoopEnable()</b>	1140
perform generic	SCSI thread initialization.....	<b>scsiThreadInit()</b>	1166
library (SCSI-2).	SCSI thread-level controller.....	<b>scsiCtrlLib</b>	256
Interface (SCSI) library	(SCSI-1). /Computer System.....	<b>scsi1Lib</b>	246
Interface (SCSI) library	(SCSI-2). /Computer System.....	<b>scsi2Lib</b>	249
commands for all devices	(SCSI-2). SCSI library common.....	<b>scsiCommonLib</b>	256
controller library	(SCSI-2). SCSI thread-level.....	<b>scsiCtrlLib</b>	256
for direct access devices	(SCSI-2). SCSI library.....	<b>scsiDirectLib</b>	257
SCSI manager library	(SCSI-2).....	<b>scsiMgrLib</b>	259
access device library	(SCSI-2). SCSI sequential.....	<b>scsiSeqLib</b>	260
initialize	SCSI-2 interface to scsiLib.....	<b>scsi2IfInit()</b>	1136
initialize SCSI-2 interface to	scsiLib.....	<b>scsi2IfInit()</b>	1136
return pointer to	SCSI_PHYS_DEV structure.....	<b>scsiPhysDevIdGet()</b>	1152
perform binary	search (ANSI).....	<b>bsearch()</b>	443
character (ANSI).	search block of memory for.....	<b>memchr()</b>	807
open directory for	searching (POSIX).....	<b>opendir()</b>	946
convert portions of	second to NTP format.....	<b>nttpsNsecToFraction()</b>	1251
convert time in	seconds into string (ANSI).....	<b>ctime()</b>	510
convert time in	seconds into string (POSIX).....	<b>ctime_r()</b>	510
secrets table. delete	secret from PPP authentication.....	<b>pppSecretDelete()</b>	994
secrets table. add	secret to PPP authentication.....	<b>pppSecretAdd()</b>	993
PPP authentication	secrets library.....	<b>pppSecretLib</b>	214
secret to PPP authentication	secrets table. add.....	<b>pppSecretAdd()</b>	993
secret from PPP authentication	secrets table. delete.....	<b>pppSecretDelete()</b>	994

	Keyword	Name	Page
display PPP authentication routine. block to block	secrets table. ....	pppSecretShow()	994
block to block (sector to	(sector to sector) transfer .....	cbioBlkCopy()	476
block) transfer routine.	sector) transfer routine. ....	cbioBlkCopy()	476
block device. read	sector(s) from SCSI .....	scsiRdSecs()	1154
block device. write	sector(s) to SCSI .....	scsiWrtSecs()	1168
numbers/ reset value of	seed used to generate random.....	srand()	1262
location of data in zbuf	segment. determine.....	zbufSegData()	1499
determine length of zbuf	segment.....	zbufSegLength()	1501
byte location. find zbuf	segment containing specified .....	zbufSegFind()	1500
into zbuf. create zbuf	segment from buffer and insert.....	zbufInsertBuf()	1497
clear	segment from CY7C604 cache. ....	cacheCy604ClearSegment()	451
get (delete and return) first	segment from module.....	moduleSegGet()	835
clear	segment from Sun-4 cache. ....	cacheSun4ClearSegment()	472
find first	segment in module.....	moduleSegFirst()	834
find next	segment in module.....	moduleSegNext()	835
get next	segment in zbuf. ....	zbufSegNext()	1501
get previous	segment in zbuf. ....	zbufSegPrev()	1502
write-protect text	segment (VxVMI). ....	vmTextProtect()	1445
wake up task pended in	select() .....	selWakeup()	1172
initialize	select facility. ....	selectInit()	1170
UNIX BSD 4.3	select library. ....	selectLib	261
get type of	select() wake-up node. ....	selWakeupType()	1174
add wake-up node to	select() wake-up list.....	selNodeAdd()	1171
find and delete node from	select() wake-up list.....	selNodeDelete()	1171
wake up all tasks in	select() wake-up list.....	selWakeupAll()	1172
initialize	select() wake-up list.....	selWakeupListInit()	1173
get number of nodes in	select() wake-up list.....	selWakeupListLen()	1173
terminate	select() wake-up list.....	selWakeupListTerm()	1174
gate type(int/trap), and gate	selector (x86). /CPU vector,.....	intVecGet2()	699
gate type(int/trap), and	selector (x86). /CPU vector,.....	intVecSet2()	703
create and initialize binary	semaphore. ....	semBCreate()	1175
create and initialize counting	semaphore. ....	semCCreate()	1176
initialize release 4.x binary	semaphore. create and .....	semCreate()	1177
delete	semaphore. ....	semDelete()	1179
event notification process for	semaphore. start .....	semEvStart()	1179
event notification process for	semaphore. stop.....	semEvStop()	1181
unlock every task pended on	semaphore. ....	semFlush()	1181
give	semaphore. ....	semGive()	1182
task IDs that are blocked on	semaphore. get list of .....	semInfo()	1183
initialize static binary	semaphore. ....	semInit()	1183
initialize mutual-exclusion	semaphore. create and .....	semMCreate()	1184
show information about	semaphore. ....	semShow()	1186
take	semaphore. ....	semTake()	1188
obtain CBIO device	semaphore. ....	cbioLock()	480
release CBIO device	semaphore. ....	cbioUnlock()	484
available/ lock (take)	semaphore, blocking if not.....	sem_wait()	1195
available. take release 4.x	semaphore, if semaphore is .....	semClear()	1177
take release 4.x semaphore, if	semaphore is available. ....	semClear()	1177
binary	semaphore library.....	semBLib	262
counting	semaphore library.....	semCLib	264

	Keyword	Name	Page
	general semaphore library.....	<b>semLib</b>	266
	mutual-exclusion semaphore library.....	<b>semMLib</b>	268
	release 4.x binary semaphore library.....	<b>semOLib</b>	271
	shared memory semaphore library (VxMP).....	<b>semSmLib</b>	274
	close named semaphore (POSIX).....	<b>sem_close()</b>	1188
	destroy unnamed semaphore (POSIX).....	<b>sem_destroy()</b>	1189
	get value of semaphore (POSIX).....	<b>sem_getvalue()</b>	1190
	initialize unnamed semaphore (POSIX).....	<b>sem_init()</b>	1191
	initialize/open named semaphore (POSIX).....	<b>sem_open()</b>	1191
	unlock (give) semaphore (POSIX).....	<b>sem_post()</b>	1193
	remove named semaphore (POSIX).....	<b>sem_unlink()</b>	1194
	unavailable/ lock (take) semaphore, returning error if.....	<b>sem_trywait()</b>	1194
	initialize POSIX semaphore show facility.....	<b>semPxShowInit()</b>	1186
	initialize semaphore show facility.....	<b>semShowInit()</b>	1187
	POSIX semaphore show library.....	<b>semPxShow</b>	273
	semaphore show routines.....	<b>semShow</b>	273
	initialize POSIX semaphore support.....	<b>semPxLibInit()</b>	1185
	library (POSIX). semaphore synchronization.....	<b>semPxLib</b>	271
	shared memory binary semaphore (VxMP). /initialize.....	<b>semBSmCreate()</b>	1175
	shared memory counting semaphore (VxMP). /initialize.....	<b>semCSmCreate()</b>	1178
	give mutual-exclusion semaphore without/.....	<b>semMGiveForce()</b>	1185
VxWorks events support for	semaphores.....	<b>semEvLib</b>	265
	set case sensitivity of volume.....	<b>dosSetVolCaseSens()</b>	563
	library (SCSI-2). SCSI sequential access device.....	<b>scsiSeqLib</b>	260
/I/O control function for	sequential access devices.....	<b>scsiSeqIoctl()</b>	1159
	create SCSI sequential device.....	<b>scsiSeqDevCreate()</b>	1158
	write file marks to SCSI sequential device.....	<b>scsiWrtFileMarks()</b>	1167
	library. tape sequential device file system.....	<b>tapeFsLib</b>	302
volume functions. associate	sequential device with tape.....	<b>tapeFsDevInit()</b>	1320
	device associated with serial channel. get SIO_CHAN.....	<b>sysSerialChanGet()</b>	1315
create VxWorks device for	serial channel.....	<b>ttyDevCreate()</b>	1405
	terminal device access to serial channels. provide.....	<b>ttyDrv</b>	326
	connect BSP serial device interrupts.....	<b>sysSerialHwInit2()</b>	1316
state. initialize BSP	serial devices to quiescent.....	<b>sysSerialHwInit()</b>	1315
	execute serializing instruction CPUID.....	<b>pentiumSerialize()</b>	973
	wide char's/ convert series of multibyte char's to.....	<b>mbstowcs()</b>	804
	multibyte char's/ convert series of wide char's to.....	<b>wcstombs()</b>	1461
File Transfer Protocol (FTP)	server.....	<b>ftpdLib</b>	113
	retrieve current DHCP server.....	<b>dhcpcServerGet()</b>	535
	display current DHCP server.....	<b>dhcpcServerShow()</b>	535
	address storage hook for server. assign permanent.....	<b>dhcpsAddressHookAdd()</b>	538
	lease storage hook for server. assign permanent.....	<b>dhcpsLeaseHookAdd()</b>	541
	log in to remote FTP server.....	<b>ftpLogin()</b>	627
	initialize NFS server.....	<b>nfsdInit()</b>	936
	get status of NFS server.....	<b>nfsdStatusGet()</b>	938
	show status of NFS server.....	<b>nfsdStatusShow()</b>	938
	set TFTP server address.....	<b>tftpPeerSet()</b>	1374
	change SNTP server broadcast settings.....	<b>sntpsConfigSet()</b>	1251
	TFTP server daemon task.....	<b>tftpdTask()</b>	1371
	address. query DNS server for host name of IP.....	<b>resolvGetHostByAddr()</b>	1075

	Keyword	Name	Page
	query DNS server for IP address of host. ....	<b>resolvGetHostByName()</b>	1076
	Network File System (NFS) server library. ....	<b>nfsdLib</b>	193
	Resolution Protocol (ARP) server library. proxy Address .....	<b>proxyArpLib</b>	215
	ICMP router discovery server library. ....	<b>rdiscLib</b>	230
	Network Time Protocol (SNTP) server library. Simple .....	<b>sntpsLib</b>	292
	server library. ....	<b>telnetdLib</b>	314
	Trivial File Transfer Protocol server library. ....	<b>tftpdLib</b>	318
	Configuration Protocol (DHCP) server library. Dynamic Host .....	<b>dhcpsLib</b>	76
	get control connection to FTP server on specified host. ....	<b>ftpHookup()</b>	626
	structures. set up DHCP server parameters and data .....	<b>dhcpsInit()</b>	540
	initialize TFTP server task .....	<b>tftpdInit()</b>	1370
	terminate FTP server task .....	<b>ftpdDelete()</b>	625
	initialize FTP server task .....	<b>ftpdInit()</b>	625
	create binding between network service and END .....	<b>muxBind()</b>	874
	detach network service from specified device .....	<b>muxUnbind()</b>	904
	initialize telnet services .....	<b>telnetdInit()</b>	1359
	initialize telnet services .....	<b>telnetdStart()</b>	1361
	close active telnet session. ....	<b>telnetdExit()</b>	1359
	initialize TFTP session. ....	<b>tftpdInit()</b>	1373
	quit TFTP session. ....	<b>tftpdQuit()</b>	1375
	command interpreter for telnet sessions. specify .....	<b>telnetdParserSet()</b>	1360
	ANSI setjmp documentation. ....	<b>ansiSetjmp</b>	16
	r1 - r14, r1-r15 for SH) (ARM, SH). /of register r0 (also .....	<b>r0()</b>	1050
	of status register (68K, SH). return contents .....	<b>sr()</b>	1262
	reduced-power mode (PowerPC, SH). place processor in .....	<b>vxPowerDown()</b>	1455
	ARM, SimSolaris, SimNT and SH). /level (68K, x86, .....	<b>intLevelSet()</b>	690
	register mach (also macl, pr) (SH). /contents of system .....	<b>mach()</b>	802
	r0 (also r1 - r14, r1-r15 for SH) (ARM, SH). /of register .....	<b>r0()</b>	1050
	/lock-out level (68K, x86, ARM, SH, SimSolaris, SimNT) .....	<b>intLockLevelGet()</b>	693
	/lock-out level (68K, x86, ARM, SH, SimSolaris, SimNT) .....	<b>intLockLevelSet()</b>	693
	/vector (68K, x86, MIPS, SH, SimSolaris, SimNT) .....	<b>intVecGet()</b>	698
	vector (trap) (68K, x86, MIPS, SH, SimSolaris, SimNT). /CPU .....	<b>intVecSet()</b>	699
	management mode (PowerPC, SH, x86). get power .....	<b>vxPowerModeGet()</b>	1456
	management mode (PowerPC, SH, x86). set power .....	<b>vxPowerModeSet()</b>	1456
	initialize SH7040 cache library. ....	<b>cacheSh7040LibInit()</b>	465
	library. Hitachi SH7040 cache management .....	<b>cacheSh7040Lib</b>	51
	initialize SH7604/SH7615 cache library. ....	<b>cacheSh7604LibInit()</b>	466
	library. Hitachi SH7604/SH7615 cache management .....	<b>cacheSh7604Lib</b>	51
	initialize SH7622 cache library. ....	<b>cacheSh7622LibInit()</b>	466
	library. SH7622 cache management .....	<b>cacheSh7622Lib</b>	52
	initialize SH7700 cache library. ....	<b>cacheSh7700LibInit()</b>	467
	library. Hitachi SH7700 cache management .....	<b>cacheSh7700Lib</b>	52
	Hitachi SH7700 MMU support library. ....	<b>mmuSh7700Lib</b>	168
	initialize SH7729 cache library. ....	<b>cacheSh7729LibInit()</b>	468
	library. Hitachi SH7729 cache management .....	<b>cacheSh7729Lib</b>	53
	initialize SH7750 cache library. ....	<b>cacheSh7750LibInit()</b>	469
	library. Hitachi SH7750 cache management .....	<b>cacheSh7750Lib</b>	53
	Hitachi SH7750 MMU support library. ....	<b>mmuSh7750Lib</b>	172
	DHCP client interface shared code library. ....	<b>dhcpcCommonLib</b>	72
	/pages to virtual space in shared global virtual mem/ .....	<b>vmGlobalMap()</b>	1438

	Keyword	Name	Page
(VxMP). create and initialize	shared memory binary semaphore .....	<b>semBSmCreate()</b>	1175
create and initialize	shared memory counting/ .....	<b>semCSmCreate()</b>	1178
library (VxMP).	shared memory management .....	<b>smMemLib</b>	281
routines (VxMP).	shared memory management show .....	<b>smMemShow</b>	284
library (VxMP).	shared memory message queue.....	<b>msgQSmLib</b>	185
(VxMP). create and initialize	shared memory message queue.....	<b>msgQSmCreate()</b>	868
(VxMP). add name to	shared memory name database .....	<b>smNameAdd()</b>	1237
show information about	shared memory network.....	<b>smNetShow()</b>	1241
VxWorks interface to	shared memory network/ .....	<b>smNetLib</b>	287
show routines.	shared memory network driver.....	<b>smNetShow</b>	288
(VxMP). look up	shared memory object by name.....	<b>smNameFind()</b>	1238
(VxMP). look up	shared memory object by value .....	<b>smNameFindByValue()</b>	1239
descriptor (VxMP). initialize	shared memory objects .....	<b>smObjInit()</b>	1244
(VxMP). attach calling CPU to	shared memory objects facility .....	<b>smObjAttach()</b>	1242
(VxMP). install	shared memory objects facility .....	<b>smObjLibInit()</b>	1245
(VxMP). initialize	shared memory objects facility .....	<b>smObjSetup()</b>	1246
(VxMP).	shared memory objects library .....	<b>smObjLib</b>	288
database library (VxMP).	shared memory objects name.....	<b>smNameLib</b>	284
database show routines/	shared memory objects name.....	<b>smNameShow</b>	286
database/ remove object from	shared memory objects name.....	<b>smNameRemove()</b>	1240
database/ show contents of	shared memory objects name.....	<b>smNameShow()</b>	1240
routines (VxMP).	shared memory objects show .....	<b>smObjShow</b>	291
display current status of	shared memory objects (VxMP).....	<b>smObjShow()</b>	1247
(VxMP). create	shared memory partition .....	<b>memPartSmCreate()</b>	818
library (VxMP).	shared memory semaphore .....	<b>semSmLib</b>	274
block of memory (VxMP). free	shared memory system partition.....	<b>smMemFree()</b>	1233
blocks and statistics/ show	shared memory system partition.....	<b>smMemShow()</b>	1236
(VxMP). add memory to	shared memory system partition.....	<b>smMemAddToPool()</b>	1231
allocate memory for array from	shared memory system partition/ .....	<b>smMemMalloc()</b>	1232
find largest free block in	shared memory system partition/ .....	<b>smMemFindMax()</b>	1233
allocate block of memory from	shared memory system partition/ .....	<b>smMemMalloc()</b>	1234
(VxMP). set debug options for	shared memory system partition.....	<b>smMemOptionsSet()</b>	1234
(VxMP). /block of memory from	shared memory system partition.....	<b>smMemRealloc()</b>	1235
start	shell .....	<b>shellInit()</b>	1213
lock access to	shell .....	<b>shellLock()</b>	1213
remote access to target	shell .....	<b>remShellLib</b>	232
machine. execute	shell command on remote .....	<b>rcmd()</b>	1057
	shell entry point .....	<b>shell()</b>	1212
	shell execution routines. ....	<b>shellLib</b>	275
display or set size of	shell history.....	<b>shellHistory()</b>	1212
display or set size of	shell history.....	<b>h()</b>	643
change	shell prompt.....	<b>shellPromptSet()</b>	1214
script. signal	shell to stop processing .....	<b>shellScriptAbort()</b>	1215
input/output/error fds. set	shell's default.....	<b>shellOrigStdSet()</b>	1214
POSIX message queue	show. ....	<b>mqPxShow</b>	180
	show AIO requests. ....	<b>aioShow()</b>	407
Register).	show all MSR (Model Specific .....	<b>pentiumMsrShow()</b>	955
specified physical device.	show BLK_DEV structures on .....	<b>scsiBlkDevShow()</b>	1139
objects name database (VxMP).	show contents of shared memory.....	<b>smNameShow()</b>	1240
configuration.	show current authentication .....	<b>ripAuthKeyShow()</b>	1091

	Keyword	Name	Page
	loaded modules.	show current status for all .....	<b>moduleShow()</b> 836
	socket interfaces.	show device information on all.....	<b>tffsShowAll()</b> 1368
	specific socket interface.	show device information on .....	<b>tffsShow()</b> 1367
	initialize POSIX semaphore	show facility. ....	<b>semPxShowInit()</b> 1186
	initialize semaphore	show facility. ....	<b>semShowInit()</b> 1187
	initialize standard I/O	show facility. ....	<b>stdioShowInit()</b> 1270
	initialize task hook	show facility. ....	<b>taskHookShowInit()</b> 1331
	initialize trigger	show facility. ....	<b>trgShowInit()</b> 1399
	initialize watchdog	show facility. ....	<b>wdShowInit()</b> 1466
	initialize DHCP	show facility. ....	<b>dhcpcShowInit()</b> 536
	initialize DSP	show facility. ....	<b>dspShowInit()</b> 566
	initialize floating-point	show facility. ....	<b>fppShowInit()</b> 604
	initialize I/O system	show facility. ....	<b>iosShowInit()</b> 712
	initialize memory partition	show facility. ....	<b>memShowInit()</b> 821
	initialize POSIX message queue	show facility. ....	<b>mqPxShowInit()</b> 838
	initialize message queue	show facility. ....	<b>msgQShowInit()</b> 867
	include virtual memory	show facility (VxVMI). ....	<b>vmShowInit()</b> 1442
	queue.	show information about message.....	<b>msgQShow()</b> 866
	semaphore.	show information about.....	<b>semShow()</b> 1186
	memory network.	show information about shared .....	<b>smNetShow()</b> 1241
	watchdog.	show information about .....	<b>wdShow()</b> 1466
	asynchronous I/O (AIO)	show library.....	<b>aioPxShow</b> 13
	POSIX semaphore	show library.....	<b>semPxShow</b> 273
	routines.	show list of task create.....	<b>taskCreateHookShow()</b> 1326
	routines.	show list of task delete .....	<b>taskDeleteHookShow()</b> 1330
	routines.	show list of task switch .....	<b>taskSwitchHookShow()</b> 1350
	Architecture) registers.	show MCA (Machine Check.....	<b>pentiumMcaShow()</b> 953
	statistics.	show network stack data pool.....	<b>netStackDataPoolShow()</b> 929
	statistics.	show network stack system pool.....	<b>netStackSysPoolShow()</b> 929
	/distributed message queue	show package (VxFusion). ....	<b>msgQDistShowInit()</b> 859
	statistics.	show partition blocks and.....	<b>memPartShow()</b> 818
	Monitoring Counters).	show PMCs (Performance.....	<b>pentiumPmcShow()</b> 970
		show pool statistics. ....	<b>netPoolShow()</b> 927
	broadcast forwarding.	show ports enabled for.....	<b>proxyPortShow()</b> 1004
		show proxy ARP networks. ....	<b>proxyNetShow()</b> 1003
	list.	show RIP interface exclusion.....	<b>ripIfExcludeListShow()</b> 1094
	trigger	show routine. ....	<b>trgShow</b> 325
	initialize task	show routine facility.....	<b>taskShowInit()</b> 1344
	dsp	show routines.....	<b>dspShow</b> 100
	floating-point	show routines.....	<b>fppShow</b> 113
	I/O system	show routines.....	<b>iosShow</b> 127
	initialize TCP	show routines.....	<b>tcpShowInit()</b> 1357
	initialize UDP	show routines.....	<b>udpShowInit()</b> 1413
	memory	show routines.....	<b>memShow</b> 161
	message queue	show routines.....	<b>msgQShow</b> 185
	and Pentium[234] specific	show routines. Pentium.....	<b>pentiumShow</b> 209
	Point-to-Point Protocol	show routines.....	<b>pppShow</b> 215
	semaphore	show routines.....	<b>semShow</b> 273
	shared memory network driver	show routines.....	<b>smNetShow</b> 288
	task hook	show routines.....	<b>taskHookShow</b> 308

	Keyword	Name	Page
	task	show routines. .... <b>taskShow</b>	312
	watchdog	show routines. .... <b>wdShow</b>	350
	initialize ICMP	show routines. .... <b>icmpShowInit()</b>	657
	initialize IGMP	show routines. .... <b>igmpShowInit()</b>	673
	initialize network	show routines. .... <b>netShowInit()</b>	928
	/message queue group	show routines (VxFusion)..... <b>msgQDistGrpShow</b>	181
	distributed message queue	show routines (VxFusion)..... <b>msgQDistShow</b>	182
	/objects interface adapter	show routines (VxFusion)..... <b>distIfShow</b>	83
	distributed name database	show routines (VxFusion)..... <b>distNameShow</b>	85
	shared memory management	show routines (VxMP). .... <b>smMemShow</b>	284
	memory objects name database	show routines (VxMP). shared..... <b>smNameShow</b>	286
	shared memory objects	show routines (VxMP). .... <b>smObjShow</b>	291
	virtual memory	show routines (VxVMI)..... <b>vmShow</b>	346
	partition blocks and/	show shared memory system..... <b>smMemShow()</b>	1236
		show state of Pty Buffers. .... <b>ptyShow()</b>	1045
	physical device.	show status information for ..... <b>scsiPhysDevShow()</b>	1153
	SCSI manager.	show status information for ..... <b>scsiMgrShow()</b>	1147
		show status of NFS server..... <b>nfsdStatusShow()</b>	938
	blocks and statistics.	show system memory partition..... <b>memShow()</b>	820
		show trigger information..... <b>trgShow()</b>	1399
	information.	show volume configuration ..... <b>cdromFsVolConfigShow()</b>	489
	handler associated with	signal. specify ..... <b>signal()</b>	1220
	wait for	signal..... <b>sigtimedwait()</b>	1224
	connect user routine to timer	signal..... <b>timer_connect()</b>	1381
	alarm clock for delivery of	signal. set..... <b>alarm()</b>	411
	initialize	signal facilities. .... <b>sigInit()</b>	1219
	initialize queued	signal facilities. .... <b>sigqueueInit()</b>	1223
	software	signal facility library..... <b>sigLib</b>	276
	(POSIX). delete	signal from signal set..... <b>sigdelset()</b>	1218
	install	signal handler. .... <b>sigvec()</b>	1226
	(POSIX). test to see if	signal is in signal set..... <b>sigismember()</b>	1220
	set	signal mask. .... <b>sigsetmask()</b>	1223
	examine calling thread's	signal mask (POSIX). /and/or..... <b>pthread_sigmask()</b>	1042
	examine and/or change	signal mask (POSIX). .... <b>sigprocmask()</b>	1221
	pulse reset	signal on SCSI bus..... <b>scsiBusReset()</b>	1139
	specify action associated with	signal (POSIX). /and/or..... <b>sigaction()</b>	1216
	suspend task until delivery of	signal (POSIX). .... <b>sigsuspend()</b>	1224
	suspend task until delivery of	signal (POSIX). .... <b>pause()</b>	950
	add signal to	signal set (POSIX). .... <b>sigaddset()</b>	1217
	delete signal from	signal set (POSIX). .... <b>sigdelset()</b>	1218
	test to see if signal is in	signal set (POSIX). .... <b>sigismember()</b>	1220
	included (POSIX). initialize	signal set with all signals..... <b>sigfillset()</b>	1219
	included (POSIX). initialize	signal set with no signals..... <b>sigemptyset()</b>	1218
	processing script.	signal shell to stop ..... <b>shellScriptAbort()</b>	1215
	(POSIX). wait for	signal to be delivered ..... <b>sigwait()</b>	1226
	send	signal to caller's task. .... <b>raise()</b>	1050
	add	signal to signal set (POSIX). .... <b>sigaddset()</b>	1217
	send queued	signal to task ..... <b>sigqueue()</b>	1222
	send	signal to task (POSIX)..... <b>kill()</b>	728
	send	signal to thread (POSIX). .... <b>pthread_kill()</b>	1030

Keyword	Name	Page
add to set of blocked signals.	<b>sigblock()</b>	1217
wait for real-time signals.	<b>sigwaitinfo()</b>	1227
retrieve set of pending signals blocked from delivery/	<b>sigpending()</b>	1221
initialize signal set with no signals included (POSIX).	<b>sigemptyset()</b>	1218
initialize signal set with all signals included (POSIX).	<b>sigfillset()</b>	1219
instrument signals (WindView).	<b>wvSigInst()</b>	1483
x86, ARM, SH, SimSolaris, SimNT). /lock-out level (68K,	<b>intLockLevelGet()</b>	693
x86, ARM, SH, SimSolaris, SimNT). /lock-out level (68K,	<b>intLockLevelSet()</b>	693
x86, MIPS, ARM, SimSolaris, SimNT). /base address (68K,	<b>intVecBaseGet()</b>	696
x86, MIPS, ARM, SimSolaris, SimNT). /base address (68K,	<b>intVecBaseSet()</b>	697
x86, MIPS, SH, SimSolaris, SimNT). /vector (68K,	<b>intVecGet()</b>	698
x86, MIPS, SH, SimSolaris, SimNT). /vector (trap) (68K,	<b>intVecSet()</b>	699
(68K, x86, ARM, SimSolaris, SimNT). /vector table.....	<b>intVecTableWriteProtect()</b>	704
(68K, x86, ARM, SimSolaris, SimNT and SH). /level.....	<b>intLevelSet()</b>	690
for C routine (68K, x86, MIPS, SimSolaris). /handler.....	<b>intHandlerCreate()</b>	688
level (68K, x86, ARM, SH, SimSolaris, SimNT). /lock-out	<b>intLockLevelGet()</b>	693
level (68K, x86, ARM, SH, SimSolaris, SimNT). /lock-out	<b>intLockLevelSet()</b>	693
address (68K, x86, MIPS, ARM, SimSolaris, SimNT). /base.....	<b>intVecBaseGet()</b>	696
address (68K, x86, MIPS, ARM, SimSolaris, SimNT). /base.....	<b>intVecBaseSet()</b>	697
vector (68K, x86, MIPS, SH, SimSolaris, SimNT). /interrupt.....	<b>intVecGet()</b>	698
(trap) (68K, x86, MIPS, SH, SimSolaris, SimNT). /vector.....	<b>intVecSet()</b>	699
vector table (68K, x86, ARM, SimSolaris, SimNT). /exception.....	<b>intVecTableWriteProtect()</b>	704
/level (68K, x86, ARM, SimSolaris, SimNT and SH).....	<b>intLevelSet()</b>	690
compute both sine and cosine	<b>sincos()</b>	1228
compute both sine and cosine	<b>sincosf()</b>	1229
compute sine (ANSI)	<b>sin()</b>	1228
compute sine (ANSI)	<b>sinf()</b>	1229
compute hyperbolic sine (ANSI)	<b>sinh()</b>	1230
compute hyperbolic sine (ANSI)	<b>sinhf()</b>	1230
compute arc sine (ANSI)	<b>asin()</b>	417
compute arc sine (ANSI)	<b>asinf()</b>	417
integer. convert single-precision value to	<b>irintf()</b>	718
subroutine. single-step, but step over	<b>so()</b>	1252
single-step task.	<b>s()</b>	1130
reset all SIO devices to quiet state.	<b>sysSerialReset()</b>	1316
with serial channel. get SIO_CHAN device associated	<b>sysSerialChanGet()</b>	1315
get current time slice (POSIX).....	<b>sched_rr_get_interval()</b>	1134
or equal to specified/ compute smallest integer greater than	<b>ceil()</b>	489
or equal to specified/ compute smallest integer greater than	<b>ceilf()</b>	490
interface routines to RIP for SNMP Agent. VxWorks.....	<b>m2RipLib</b>	147
MIB-II ICMP-group API for SNMP Agents.	<b>m2IcmpLib</b>	139
MIB-II interface-group API for SNMP agents.....	<b>m2IfLib</b>	139
MIB-II IP-group API for SNMP agents.....	<b>m2IpLib</b>	142
MIB-II API library for SNMP agents.....	<b>m2Lib</b>	144
MIB-II system-group API for SNMP agents.....	<b>m2SysLib</b>	148
MIB-II TCP-group API for SNMP agents.....	<b>m2TcpLib</b>	150
MIB-II UDP-group API for SNMP agents.....	<b>m2UdpLib</b>	152
initialize SNMP MIB-2 library.....	<b>m2Init()</b>	786
inform SCSI that hardware snooping of caches is/	<b>scsiCacheSnoopDisable()</b>	1140
inform SCSI that hardware snooping of caches is enabled.	<b>scsiCacheSnoopEnable()</b>	1140

	Keyword	Name	Page
Simple Network Time Protocol	(SNTP) client library.....	<b>sntpcLib</b>	291
settings. change	SNTP server broadcast.....	<b>sntpsConfigSet()</b>	1251
Simple Network Time Protocol	(SNTP) server library.....	<b>sntpsLib</b>	292
receive data from	socket.....	<b>recv()</b>	1067
receive message from	socket.....	<b>recvfrom()</b>	1068
receive message from	socket.....	<b>recvmsg()</b>	1069
send data to	socket.....	<b>send()</b>	1196
send message to	socket.....	<b>sendmsg()</b>	1197
send message to	socket.....	<b>sendto()</b>	1198
open	socket.....	<b>socket()</b>	1252
user data and send it to TCP	socket. create zbuf from.....	<b>zbufSockBufSend()</b>	1502
message and send it to UDP	socket. create zbuf from user.....	<b>zbufSockBufSendto()</b>	1503
receive data in zbuf from TCP	socket.....	<b>zbufSockRecv()</b>	1505
message in zbuf from UDP	socket. receive.....	<b>zbufSockRecvfrom()</b>	1506
send zbuf data to TCP	socket.....	<b>zbufSockSend()</b>	1507
send zbuf message to UDP	socket.....	<b>zbufSockSendto()</b>	1508
accept connection from	socket.....	<b>accept()</b>	404
bind name to	socket.....	<b>bind()</b>	430
initiate connection to	socket.....	<b>connect()</b>	497
enable connections to	socket.....	<b>listen()</b>	736
route message. extract	socket address pointers from.....	<b>ripAddrXtract()</b>	1083
specified duration. attempt	socket connection within.....	<b>connectWithTimeout()</b>	498
device information on specific	socket interface. show.....	<b>tffsShow()</b>	1367
initialize zbuf	socket interface library.....	<b>zbufSockLibInit()</b>	1505
zbuf	socket interface library.....	<b>zbufSockLib</b>	361
show device information on all	socket interfaces.....	<b>tffsShowAll()</b>	1368
generic	socket library.....	<b>sockLib</b>	293
get	socket name.....	<b>getsockname()</b>	638
set	socket options.....	<b>setsockopt()</b>	1203
get	socket options.....	<b>getsockopt()</b>	639
bind	socket to privileged IP port.....	<b>bindresvport()</b>	431
close	socket upload path library.....	<b>wvSockUploadPathLib</b>	357
write to	socket upload path (Windview).....	<b>sockUploadPathClose()</b>	1253
upload path to host using	socket (Windview). establish.....	<b>sockUploadPathWrite()</b>	1254
upload path to host using TSFS	socket (Windview). open.....	<b>tsfsUploadPathCreate()</b>	1402
bound to it. open	socket with privileged port.....	<b>rresvport()</b>	1125
for Internet protocol	sockets. /active connections.....	<b>inetstatShow()</b>	681
support. initialize	software floating-point math.....	<b>mathSoftInit()</b>	803
library.	software signal facility.....	<b>sigLib</b>	276
/disk driver (VxSim for	Solaris and VxSim for HP).....	<b>unixDrv</b>	332
sort array of objects (ANSI).	.....	<b>qsort()</b>	1049
current time from remote	source. retrieve.....	<b>sntpcTimeGet()</b>	1249
(use <b>unld()</b> to reclaim	space). /module ID information.....	<b>moduleDelete()</b>	830
/is printable, including	space character (ANSI).....	<b>isprint()</b>	723
allocate	space for array (ANSI). .....	<b>calloc()</b>	476
map physical pages to virtual	space in shared global virtual/.....	<b>vmGlobalMap()</b>	1438
(VxVMI). map physical	space into virtual space.....	<b>vmMap()</b>	1440
physical space into virtual	space (VxVMI). map.....	<b>vmMap()</b>	1440
spawn task. ....	.....	<b>taskSpawn()</b>	1345

	Keyword	Name	Page
	periodically.	spawn task to call function .....	<b>period()</b> 975
	repeatedly.	spawn task to call function .....	<b>repeat()</b> 1072
	parameters.	spawn task with default .....	<b>sp()</b> 1255
	of failed attempts to take	spin-lock (VxMP). /logging.....	<b>smObjTimeoutLogEnable()</b> 1248
	zbufs.	split zbuf into two separate .....	<b>zbufSplit()</b> 1509
		spy CPU activity library. ....	<b>spyLib</b> 294
	stop	spying and reporting. ....	<b>spyStop()</b> 1260
	compute non-negative	square root (ANSI). ....	<b>sqrt()</b> 1261
	compute non-negative	square root (ANSI). ....	<b>sqrtf()</b> 1261
	routine for TCP/IP network	stack. generic attach .....	<b>ipAttach()</b> 714
	routine for TCP/IP network	stack. generic detach.....	<b>ipDetach()</b> 714
	initialize task with	stack at specified address.....	<b>taskInit()</b> 1334
	show network	stack data pool statistics.....	<b>netStackDataPoolShow()</b> 929
	pop cleanup routine off top of	stack (POSIX). ....	<b>pthread_cleanup_pop()</b> 1019
	pushes routine onto cleanup	stack (POSIX). ....	<b>pthread_cleanup_push()</b> 1019
	(WindView). set priority and	stack size of tWVUpload task.....	<b>wvUploadTaskConfig()</b> 1488
	show network	stack system pool statistics. ....	<b>netStackSysPoolShow()</b> 929
	display	stack trace of task. ....	<b>tt()</b> 1404
	print summary of each task's	stack usage. ....	<b>checkStack()</b> 491
	enable or disable interrupt	stack usage (x86).....	<b>intStackEnable()</b> 695
	attribute in thread/ get	stack value of stacksize.....	<b>pthread_attr_getstacksize()</b> 1012
	thread/ get value of	stackaddr attribute from .....	<b>pthread_attr_getstackaddr()</b> 1011
	attributes object/ set	stackaddr attribute in thread .....	<b>pthread_attr_setstackaddr()</b> 1017
	attributes/ get stack value of	stacksize attribute in thread.....	<b>pthread_attr_getstacksize()</b> 1012
	attributes object/ set	stacksize attribute in thread.....	<b>pthread_attr_setstacksize()</b> 1018
	SCSI device. issue	START_STOP_UNIT command to.....	<b>scsiStartStopUnit()</b> 1162
	serial devices to quiescent	state. initialize BSP .....	<b>sysSerialHwInit()</b> 1315
	reset all SIO devices to quiet	state. ....	<b>sysSerialReset()</b> 1316
	set TCP connection to closed	state. ....	<b>m2TcpConnEntrySet()</b> 798
	(POSIX). set cancellation	state for calling thread .....	<b>pthread_setcancelstate()</b> 1039
	initialize global	state for MUX.....	<b>mutexLibInit()</b> 881
	context. determine if current	state is in interrupt or task .....	<b>intContext()</b> 686
	send event to SCSI controller	state machine. ....	<b>scsiMgrCtrlEvent()</b> 1146
	send event to thread	state machine. ....	<b>scsiMgrThreadEvent()</b> 1148
	virtual memory. change	state of block of.....	<b>vmBaseStateSet()</b> 1433
	virtual memory/ change	state of block of.....	<b>vmStateSet()</b> 1444
	entry to UP or DOWN. set	state of MIB-II interface .....	<b>m2IfTblEntrySet()</b> 784
	virtual memory (VxVMI). get	state of page of.....	<b>vmStateGet()</b> 1443
	show	state of Pty Buffers. ....	<b>ptyShow()</b> 1045
	RIP. add hook to install	static and non-RIP routes into .....	<b>ripRouteHookAdd()</b> 1099
	initialize	static binary semaphore. ....	<b>semInit()</b> 1183
	strategy (C++). change C++	static constructor calling.....	<b>cplusplusXtorSet()</b> 507
	call	static constructors (C++). ....	<b>cplusplusCtors()</b> 503
	call all linked	static constructors (C++). ....	<b>cplusplusCtorsLink()</b> 504
	call	static destructors (C++). ....	<b>cplusplusDtors()</b> 505
	call all linked	static destructors (C++). ....	<b>cplusplusDtorsLink()</b> 506
	display routing	statistics.....	<b>routeStatShow()</b> 1122
	display IP	statistics.....	<b>ipStatShow()</b> 717
	report mbuf	statistics.....	<b>mbufShow()</b> 805
	show partition blocks and	statistics.....	<b>memPartShow()</b> 818

	Keyword	Name	Page
memory partition blocks and	statistics. show system .....	<b>memShow()</b>	820
show pool	statistics. ....	<b>netPoolShow()</b>	927
show network stack data pool	statistics. ....	<b>netStackDataPoolShow()</b>	929
show network stack system pool	statistics. ....	<b>netStackSysPoolShow()</b>	929
get PPP link	statistics. ....	<b>pppstatGet()</b>	995
display PPP link	statistics. ....	<b>pppstatShow()</b>	995
display	statistics for ICMP. ....	<b>icmpstatShow()</b>	658
display	statistics for IGMP. ....	<b>igmpstatShow()</b>	674
display all	statistics for TCP protocol. ....	<b>tcpstatShow()</b>	1358
display	statistics for UDP protocol. ....	<b>udpstatShow()</b>	1413
system partition blocks and	statistics (VxMP). /memory .....	<b>smMemShow()</b>	1236
<b>rawFsLib</b> of change in ready	status. notify .....	<b>rawFsReadyChange()</b>	1056
rt11Fs of change in ready	status. notify .....	<b>rt11FsReadyChange()</b>	1128
<b>tapeFsLib</b> of change in ready	status. notify .....	<b>tapeFsReadyChange()</b>	1321
get task's	status as string. ....	<b>taskStatusString()</b>	1347
show current	status for all loaded modules. ....	<b>moduleShow()</b>	836
get TFTP	status information. ....	<b>tftpInfoShow()</b>	1372
get PPP link	status information. ....	<b>pppInfoGet()</b>	984
display PPP link	status information. ....	<b>pppInfoShow()</b>	985
physical device. show	status information for. ....	<b>scsiPhysDevShow()</b>	1153
manager. show	status information for SCSI .....	<b>scsiMgrShow()</b>	1147
get file	status information (POSIX). ....	<b>fstat()</b>	619
get file	status information (POSIX). ....	<b>fstatfs()</b>	620
pathname (POSIX). get file	status information using. ....	<b>stat()</b>	1268
pathname (POSIX). get file	status information using. ....	<b>statfs()</b>	1268
error	status library. ....	<b>errnoLib</b>	102
operation/ retrieve error	status of asynchronous I/O. ....	<b>aio_error()</b>	408
operation/ retrieve return	status of asynchronous I/O. ....	<b>aio_return()</b>	409
determine ready	status of CBIO device. ....	<b>cbioRdyChgdGet()</b>	482
force change in ready	status of CBIO device. ....	<b>cbioRdyChgdSet()</b>	483
get	status of NFS server. ....	<b>nfsdStatusGet()</b>	938
show	status of NFS server. ....	<b>nfsdStatusShow()</b>	938
modify	status of relationship. ....	<b>m2IfStackEntrySet()</b>	781
objects/ display current	status of shared memory. ....	<b>smObjShow()</b>	1247
x86). set task	status register (68K, MIPS). ....	<b>taskSRSet()</b>	1347
return contents of	status register (68K, SH). ....	<b>sr()</b>	1262
contents of current processor	status register (ARM). return .....	<b>cpsr()</b>	508
initialize default task	status register (MIPS). ....	<b>taskSRInit()</b>	1346
read contents of	status register (MIPS). ....	<b>intSRGet()</b>	694
update contents of	status register (MIPS). ....	<b>intSRSet()</b>	694
return contents of	status register (x86/SimNT). ....	<b>eflags()</b>	568
definition of specified error	status value. print .....	<b>printErrno()</b>	996
get error	status value of calling task. ....	<b>errnoGet()</b>	571
set error	status value of calling task. ....	<b>errnoSet()</b>	573
task. get error	status value of specified. ....	<b>errnoOfTaskGet()</b>	572
task. set error	status value of specified. ....	<b>errnoOfTaskSet()</b>	572
ANSI	stdarg documentation. ....	<b>ansiStdarg</b>	17
copy in (or	stdin) to out (or stdout). ....	<b>copy()</b>	498
ANSI	stdio documentation. ....	<b>ansiStdio</b>	18
ANSI	stdlib documentation. ....	<b>ansiStdlib</b>	22

	Keyword	Name	Page
copy in (or stdin) to out (or	stdout).....	copy()	498
assign permanent address	storage hook for server.....	dhcpsAddressHookAdd()	538
assign permanent lease	storage hook for server.....	dhcpsLeaseHookAdd()	541
delete lease data	storage routine.....	dhcpcCacheHookDelete()	524
C++ static constructor calling	strategy (C++). change.....	cplusXtorSet()	507
write word (32-bit integer) to	stream.....	putw()	1047
specify buffering for	stream.....	setbuffer()	1200
word (32-bit integer) from	stream. read next.....	getw()	640
string to standard error	stream. write formatted.....	printErr()	996
write character to	stream (ANSI).....	putc()	1045
character to standard output	stream (ANSI). write.....	putchar()	1046
string to standard output	stream (ANSI). write.....	puts()	1047
characters from standard input	stream (ANSI). /and convert.....	scanf()	1130
specify buffering for	stream (ANSI).....	setbuf()	1200
specify buffering for	stream (ANSI).....	setvbuf()	1211
push character back into input	stream (ANSI).....	ungetc()	1413
write formatted string to	stream (ANSI).....	vfprintf()	1431
and error flags for	stream (ANSI). /end-of-file.....	clearerr()	493
close	stream (ANSI).....	fclose()	587
test end-of-file indicator for	stream (ANSI).....	feof()	588
flush	stream (ANSI).....	fflush()	589
return next character from	stream (ANSI).....	fgetc()	590
of file position indicator for	stream (ANSI). /current value.....	fgetpos()	590
number of characters from	stream (ANSI). read specified.....	fgets()	591
write formatted string to	stream (ANSI).....	fprintf()	606
write character to	stream (ANSI).....	fputc()	610
write string to	stream (ANSI).....	fputs()	611
and convert characters from	stream (ANSI). read.....	fscanf()	614
file position indicator for	stream (ANSI). set.....	fseek()	618
file position indicator for	stream (ANSI). set.....	fsetpos()	619
of file position indicator for	stream (ANSI). /current value.....	ftell()	620
return next character from	stream (ANSI).....	getc()	635
character from standard input	stream (ANSI). return next.....	getchar()	635
characters from standard input	stream (ANSI). read.....	gets()	638
string to standard output	stream (ANSI). /formatted.....	printf()	997
transfer file via TFTP using	stream interface.....	tftpXfer()	1376
return fd for	stream (POSIX).....	fileno()	591
copy from/to specified	streams.....	copyStreams()	499
occurrence of character in	string. find last.....	rindex()	1083
get task's status as	string.....	taskStatusString()	1347
occurrence of character in	string. find first.....	index()	674
return kernel revision	string.....	kernelVersion()	727
change login	string.....	loginStringSet()	754
convert characters from ASCII	string (ANSI). read and.....	sscanf()	1263
occurrence of character in	string (ANSI). find first.....	strchr()	1271
map error number to error	string (ANSI).....	strerror()	1274
time into formatted	string (ANSI). /broken-down.....	strftime()	1275
determine length of	string (ANSI).....	strlen()	1277
occurrence of character in	string (ANSI). find last.....	strrchr()	1279
occurrence of substring in	string (ANSI). find first.....	strstr()	1280

	Keyword	Name	Page
convert broken-down time into	string (ANSI). .....	<b>asctime()</b>	416
convert time in seconds into	string (ANSI). .....	<b>ctime()</b>	510
ANSI	string documentation. ....	<b>ansiString</b>	24
argument list to buffer/ write	string formatted with variable .....	<b>vsprintf()</b>	1446
argument list to fd. write	string formatted with variable .....	<b>vfdprintf()</b>	1430
argument list to/ write	string formatted with variable .....	<b>vprintf()</b>	1446
read	string from file. ....	<b>fioRdString()</b>	595
break down	string into tokens (ANSI). ....	<b>strtok()</b>	1282
(POSIX). break down	string into tokens (reentrant). ....	<b>strtok_r()</b>	1283
character from given/ return	string length up to first .....	<b>strcspn()</b>	1273
character not in given/ return	string length up to first .....	<b>strspn()</b>	1280
find device using	string name. ....	<b>endFindByName()</b>	569
set/ find first occurrence in	string of character from given. ....	<b>strpbrk()</b>	1279
map error number to error	string (POSIX). ....	<b>strerror_r()</b>	1274
convert broken-down time into	string (POSIX). ....	<b>asctime_r()</b>	416
convert time in seconds into	string (POSIX). ....	<b>ctime_r()</b>	510
Internet network number from	string to address. convert .....	<b>inet_network()</b>	679
concatenate one	string to another (ANSI). ....	<b>strcat()</b>	1271
copy one	string to another (ANSI). ....	<b>strcpy()</b>	1273
/characters from one	string to another (ANSI). ....	<b>strncat()</b>	1277
copy characters from one	string to another (ANSI). ....	<b>strncpy()</b>	1278
write formatted	string to buffer (ANSI). ....	<b>sprintf()</b>	1256
(Unimplemented) (ANSI). pass	string to command processor. ....	<b>system()</b>	1317
convert initial portion of	string to double (ANSI). ....	<b>strtod()</b>	1281
convert	string to double (ANSI). ....	<b>atof()</b>	421
write formatted	string to fd. ....	<b>fdprintf()</b>	588
post user event	string to host tools. ....	<b>wdbUserEvtPost()</b>	1463
convert	string to int (ANSI). ....	<b>atoi()</b>	422
convert	string to long (ANSI). ....	<b>atol()</b>	422
convert	string to long integer (ANSI). ....	<b>strtol()</b>	1284
stream. write formatted	string to standard error .....	<b>printErr()</b>	996
stream (ANSI). write	string to standard output. ....	<b>puts()</b>	1047
stream/ write formatted	string to standard output. ....	<b>printf()</b>	997
write formatted	string to stream (ANSI). ....	<b>vfprintf()</b>	1431
write formatted	string to stream (ANSI). ....	<b>fprintf()</b>	606
write	string to stream (ANSI). ....	<b>fputs()</b>	611
integer (ANSI). convert	string to unsigned long .....	<b>strtoul()</b>	1285
scaling hashing function for	strings. iterative. ....	<b>hashFuncIterScale()</b>	643
first n characters of two	strings (ANSI). compare .....	<b>strncmp()</b>	1278
LC_COLLATE/ compare two	strings as appropriate to .....	<b>strcoll()</b>	1272
(ANSI). compare two	strings lexicographically .....	<b>strcmp()</b>	1272
compare keys based on	strings they point to. ....	<b>hashKeyStrCmp()</b>	645
of directory and any of	subdirectories. list contents .....	<b>lsr()</b>	761
listing of directory and all	subdirectories contents. /long .....	<b>llr()</b>	738
update relationship between	sub-layers. ....	<b>m2IfStackTblUpdate()</b>	782
extract	sublist from list. ....	<b>lstExtract()</b>	763
define	subnet for network interface. ....	<b>ifMaskSet()</b>	668
interface. get	subnet mask for network .....	<b>ifMaskGet()</b>	668
find first occurrence of	substring in string (ANSI). ....	<b>strstr()</b>	1280
create TrueFFS block device	suitable for use with dosFs. ....	<b>tffsDevCreate()</b>	1363

	Keyword	Name	Page
clear specific context from	Sun-4 cache.....	<b>cacheSun4ClearContext()</b>	470
clear line from	Sun-4 cache.....	<b>cacheSun4ClearLine()</b>	471
clear page from	Sun-4 cache.....	<b>cacheSun4ClearPage()</b>	471
clear segment from	Sun-4 cache.....	<b>cacheSun4ClearSegment()</b>	472
initialize	Sun-4 cache library.....	<b>cacheSun4LibInit()</b>	472
library.	Sun-4 cache management.....	<b>cacheSun4Lib</b>	54
(MC68060). disable	superscalar dispatch .....	<b>vxSSDisable()</b>	1458
(MC68060). enable	superscalar dispatch .....	<b>vxSSEnable()</b>	1458
time interval elapses/	suspend current task until .....	<b>nanosleep()</b>	906
	suspend system.....	<b>wdbSystemSuspend()</b>	1462
	suspend task.....	<b>taskSuspend()</b>	1348
	suspend task.....	<b>ts()</b>	1401
signal (POSIX).	suspend task until delivery of.....	<b>sigsuspend()</b>	1224
signal (POSIX).	suspend task until delivery of.....	<b>pause()</b>	950
check if task is	suspended.....	<b>taskIsSuspended()</b>	1335
	swap buffers.....	<b>bswap()</b>	444
	swap bytes.....	<b>swab()</b>	1287
are not necessarily aligned.	swap bytes with buffers that .....	<b>uswab()</b>	1426
to be called at every task	switch. add routine .....	<b>taskSwitchHookAdd()</b>	1349
delete previously added task	switch routine.....	<b>taskSwitchHookDelete()</b>	1350
show list of task	switch routines.....	<b>taskSwitchHookShow()</b>	1350
set	symbolic name of this machine.....	<b>sethostname()</b>	1201
get	symbolic name of this machine.....	<b>gethostname()</b>	637
of specified psr value,	symbolically (ARM). /meaning .....	<b>psrShow()</b>	1006
list	symbols.....	<b>lkup()</b>	737
specified value. list	symbols whose values are near.....	<b>lkAddr()</b>	736
host/target symbol table	synchronization. initialize.....	<b>symSyncLibInit()</b>	1295
host/target symbol table	synchronization.....	<b>symSyncLib</b>	297
(POSIX). semaphore	synchronization library.....	<b>semPxLib</b>	271
coherency.	synchronize caches for data.....	<b>scsiCacheSynchronize()</b>	1141
data caches.	synchronize instruction and .....	<b>cacheTextUpdate()</b>	473
facilities. display	synopsis of execution timer .....	<b>timexHelp()</b>	1387
functions. print	synopsis of I/O utility .....	<b>ioHelp()</b>	707
print	synopsis of network routines.....	<b>netHelp()</b>	914
print	synopsis of selected routines.....	<b>help()</b>	651
	system-dependent library.....	<b>sysLib</b>	299
agents. MIB-II	system-group API for SNMP.....	<b>m2SysLib</b>	148
get	system-group MIB-II variables.....	<b>m2SysGroupInfoGet()</b>	796
to new values. set	system-group MIB-II variables.....	<b>m2SysGroupInfoSet()</b>	796
initialize MIB-II	system-group routines.....	<b>m2SysInit()</b>	797
list all	system-known devices.....	<b>devs()</b>	519
not available (POSIX). lock	(take) semaphore, blocking if .....	<b>sem_wait()</b>	1195
error if unavailable/ lock	(take) semaphore, returning .....	<b>sem_trywait()</b>	1194
compute	tangent (ANSI).....	<b>tan()</b>	1318
compute	tangent (ANSI).....	<b>tanf()</b>	1318
compute hyperbolic	tangent (ANSI).....	<b>tanh()</b>	1319
compute hyperbolic	tangent (ANSI).....	<b>tanhf()</b>	1319
compute arc	tangent (ANSI).....	<b>atan()</b>	418
compute arc	tangent (ANSI).....	<b>atanf()</b>	420
compute arc	tangent of y/x (ANSI).....	<b>atan2()</b>	419

	Keyword	Name	Page
compute arc tangent of y/x (ANSI) .....		<b>atan2f()</b>	420
all files from tar formatted	tape. extract.....	<b>tarExtract()</b>	1323
all contents of tar formatted	tape. display.....	<b>tarToc()</b>	1324
read bytes or blocks from SCSI	tape device.....	<b>scsiRdTape()</b>	1154
command to SCSI	tape device. issue MODE_SELECT .....	<b>scsiTapeModeSelect()</b>	1163
MODE_SENSE command to SCSI	tape device. issue .....	<b>scsiTapeModeSense()</b>	1163
write data to SCSI	tape device.....	<b>scsiWrtTape()</b>	1168
disable	tape device volume.....	<b>tapeFsVolUnmount()</b>	1322
archive named file/dir onto	tape in tar format.....	<b>tarArchive()</b>	1322
SCSI device. move	tape on specified physical.....	<b>scsiSpace()</b>	1161
system library.	tape sequential device file.....	<b>tapeFsLib</b>	302
/sequential device with	tape volume functions.....	<b>tapeFsDevInit()</b>	1320
initialize	tape volume library.....	<b>tapeFsInit()</b>	1321
status. notify	<b>tapeFsLib</b> of change in ready.....	<b>tapeFsReadyChange()</b>	1321
UNIX	tar compatible library.....	<b>tarLib</b>	306
named file/dir onto tape in	tar format. archive.....	<b>tarArchive()</b>	1322
extract all files from	tar formatted tape.....	<b>tarExtract()</b>	1323
display all contents of	tar formatted tape.....	<b>tarToc()</b>	1324
SCSI message received from	target. handle complete.....	<b>scsiMsgInComplete()</b>	1150
options for specified SCSI	target. display .....	<b>scsiTargetOptionsShow()</b>	1165
using TSFS.	target host connection library .....	<b>wvTsfsUploadPathLib</b>	358
remote access to	target shell.....	<b>remShellLib</b>	232
options for one or all SCSI	targets. get.....	<b>scsiTargetOptionsGet()</b>	1164
options for one or all SCSI	targets. set .....	<b>scsiTargetOptionsSet()</b>	1164
send signal to caller's	task.....	<b>raise()</b>	1050
single-step	task.....	<b>s()</b>	1130
send queued signal to	task.....	<b>sigqueue()</b>	1222
FILE of current	task. /input/output/error .....	<b>stdioFp()</b>	1269
delete	task.....	<b>taskDelete()</b>	1327
get task ID of running	task.....	<b>taskIdSelf()</b>	1332
verify existence of	task.....	<b>taskIdVerify()</b>	1333
get information about	task.....	<b>taskInfoGet()</b>	1333
examine priority of	task.....	<b>taskPriorityGet()</b>	1339
change priority of	task.....	<b>taskPrioritySet()</b>	1339
restart	task.....	<b>taskRestart()</b>	1341
resume	task.....	<b>taskResume()</b>	1342
spawn	task.....	<b>taskSpawn()</b>	1345
suspend	task.....	<b>taskSuspend()</b>	1348
add task variable to	task.....	<b>taskVarAdd()</b>	1352
remove task variable from	task.....	<b>taskVarDelete()</b>	1354
get list of task variables of	task.....	<b>taskVarInfo()</b>	1355
delete	task.....	<b>td()</b>	1358
initialize TFTP server	task.....	<b>tftpdInit()</b>	1370
TFTP server daemon	task.....	<b>tftpdTask()</b>	1371
resume	task.....	<b>tr()</b>	1393
suspend	task.....	<b>ts()</b>	1401
display stack trace of	task.....	<b>tt()</b>	1404
display environment for	task.....	<b>envShow()</b>	571
error status value of calling	task. get.....	<b>errnoGet()</b>	571
status value of specified	task. get error.....	<b>errnoOfTaskGet()</b>	572

	Keyword	Name	Page
status value of specified	task. set error.....	<b>errnoOfTaskSet()</b>	572
error status value of calling	task. set.....	<b>errnoSet()</b>	573
clear all events for current	task. ....	<b>eventClear()</b>	575
floating-point registers of	task. set.....	<b>fppTaskRegsSet()</b>	605
terminate FTP server	task. ....	<b>ftpdDelete()</b>	625
initialize FTP server	task. ....	<b>ftpdInit()</b>	625
message-logging support	task. ....	<b>logTask()</b>	758
initialize and start MUX poll	task. ....	<b>muxPollStart()</b>	891
get delay on polling	task. ....	<b>muxTaskDelayGet()</b>	893
inter-cycle delay on polling	task. set.....	<b>muxTaskDelaySet()</b>	894
start collecting	task activity data.....	<b>spyClkStart()</b>	1257
stop collecting	task activity data.....	<b>spyClkStop()</b>	1258
display	task activity data.....	<b>spyReport()</b>	1259
begin periodic	task activity reports.....	<b>spy()</b>	1257
run periodic	task activity reports.....	<b>spyTask()</b>	1260
reset trigger work queue	task and queue.....	<b>trgWorkQReset()</b>	1400
exit	task (ANSI).....	<b>exit()</b>	584
state is in interrupt or	task context. /if current.....	<b>intContext()</b>	686
ID. get	task control block for task.....	<b>taskTcb()</b>	1351
package. initialize	task CPU utilization tool.....	<b>spyLibInit()</b>	1259
routine to be called at every	task create. add.....	<b>taskCreateHookAdd()</b>	1325
delete previously added	task create routine.....	<b>taskCreateHookDelete()</b>	1326
show list of	task create routines.....	<b>taskCreateHookShow()</b>	1326
routine to be called at every	task delete. add.....	<b>taskDeleteHookAdd()</b>	1329
delete previously added	task delete routine.....	<b>taskDeleteHookDelete()</b>	1329
show list of	task delete routines.....	<b>taskDeleteHookShow()</b>	1330
network	task entry point.....	<b>netTask()</b>	930
delay	task from executing.....	<b>taskDelay()</b>	1327
initialize	task hook facilities.....	<b>taskHookInit()</b>	1330
	task hook library.....	<b>taskHookLib</b>	307
initialize	task hook show facility.....	<b>taskHookShowInit()</b>	1331
	task hook show routines.....	<b>taskHookShow</b>	308
set default	task ID.....	<b>taskIdDefault()</b>	1331
get name associated with	task ID.....	<b>taskIdName()</b>	1336
get task control block for	task ID.....	<b>taskTcb()</b>	1351
name. look up	task ID associated with task.....	<b>taskIdNameToId()</b>	1337
get	task ID of running task.....	<b>taskIdSelf()</b>	1332
get list of active	task IDs.....	<b>taskIdListGet()</b>	1332
semaphore. get list of	task IDs that are blocked on.....	<b>semInfo()</b>	1183
display	task information from TCBs.....	<b>taskShow()</b>	1343
	task information library.....	<b>taskInfo</b>	309
check if	task is ready to run.....	<b>taskIsReady()</b>	1335
check if	task is suspended.....	<b>taskIsSuspended()</b>	1335
	task management library.....	<b>taskLib</b>	310
architecture-specific	task management routines.....	<b>taskArchLib</b>	307
display	task monitoring help menu.....	<b>spyHelp()</b>	1258
up task ID associated with	task name. look.....	<b>taskIdNameToId()</b>	1337
preserve extra copy of	task name events (WindView).....	<b>wvTaskNamesPreserve()</b>	1484
upload preserved	task name events (WindView).....	<b>wvTaskNamesUpload()</b>	1485
examine	task options.....	<b>taskOptionsGet()</b>	1337

	Keyword	Name	Page
	change task options.....	<b>taskOptionsSet()</b>	1338
	wake up task pending in <b>select()</b> .....	<b>selWakeup()</b>	1172
	unblock every task pending on semaphore.....	<b>semFlush()</b>	1181
	parameters for specified task (POSIX). get scheduling.....	<b>sched_getparam()</b>	1132
	send signal to task (POSIX).....	<b>kill()</b>	728
	get content of TASK register (x86).....	<b>vxTssGet()</b>	1460
	set value to TASK register (x86).....	<b>vxTssSet()</b>	1460
	disable task rescheduling.....	<b>taskLock()</b>	1336
	enable task rescheduling.....	<b>taskUnlock()</b>	1351
	make calling task safe from deletion.....	<b>taskSafe()</b>	1342
	initialize task show routine facility.....	<b>taskShowInit()</b>	1344
	task show routines.....	<b>taskShow</b>	312
	get fd for task standard/.....	<b>ioTaskStdGet()</b>	713
	set fd for task standard/.....	<b>ioTaskStdSet()</b>	713
	MIPS, x86). set task status register (68K,.....	<b>taskSRSet()</b>	1347
	initialize default task status register (MIPS).....	<b>taskSRInit()</b>	1346
	routine to be called at every task switch. add.....	<b>taskSwitchHookAdd()</b>	1349
	delete previously added task switch routine.....	<b>taskSwitchHookDelete()</b>	1350
	show list of task switch routines.....	<b>taskSwitchHookShow()</b>	1350
	floating-point registers from task TCB. get.....	<b>fppTaskRegsGet()</b>	605
	initialized. activate task that has been.....	<b>taskActivate()</b>	1325
	on queue (POSIX). notify task that message is available.....	<b>mq_notify()</b>	840
	periodically. spawn task to call function.....	<b>period()</b>	975
	repeatedly. spawn task to call function.....	<b>repeat()</b>	1072
	make calling task unsafe from deletion.....	<b>taskUnsafe()</b>	1352
	(POSIX). suspend task until delivery of signal.....	<b>sigsuspend()</b>	1224
	(POSIX). suspend task until delivery of signal.....	<b>pause()</b>	950
	elapses/ suspend current task until time interval.....	<b>nanosleep()</b>	906
	get value of task variable.....	<b>taskVarGet()</b>	1354
	set value of task variable.....	<b>taskVarSet()</b>	1356
	remove task variable from task.....	<b>taskVarDelete()</b>	1354
	add task variable to task.....	<b>taskVarAdd()</b>	1352
	initialize task variables facility.....	<b>taskVarInit()</b>	1355
	get list of task variables of task.....	<b>taskVarInfo()</b>	1355
	library. task variables support.....	<b>taskVarLib</b>	313
	and stack size of tWVUpload task (WindView). set priority.....	<b>wvUploadTaskConfig()</b>	1488
	spawn task with default parameters.....	<b>sp()</b>	1255
	address. initialize task with stack at specified.....	<b>taskInit()</b>	1334
	delete task without restriction.....	<b>taskDeleteForce()</b>	1328
	handle task-level exceptions.....	<b>excTask()</b>	582
	device. do task-level read for tty.....	<b>tyRead()</b>	1411
	device. do task-level write for tty.....	<b>tyWrite()</b>	1412
	initialize task's access to RPC package.....	<b>rpcTaskInit()</b>	1124
	print contents of task's DSP registers.....	<b>dspTaskRegsShow()</b>	566
	registers. print contents of task's floating-point.....	<b>fppTaskRegsShow()</b>	606
	wake-up list. wake up all tasks in <b>select()</b> .....	<b>selWakeupAll()</b>	1172
	set task's priority (POSIX).....	<b>sched_setparam()</b>	1134
	set task's registers.....	<b>taskRegsSet()</b>	1340
	display contents of task's registers.....	<b>taskRegsShow()</b>	1341
	get task's registers from TCB.....	<b>taskRegsGet()</b>	1340

	Keyword	Name	Page
	print summary of each task's stack usage.	<code>checkStack()</code>	491
	get task's status as string.	<code>taskStatusString()</code>	1347
	complete information from task's TCB. print	<code>ti()</code>	1378
	print summary of each task's TCB.	<code>i()</code>	656
	telnetd. report whether tasks were pre-started by ...	<code>telnetdStaticTaskInitializationGet()</code>	1362
	get task's registers from TCB.	<code>taskRegsGet()</code>	1340
	information from task's TCB. print complete	<code>ti()</code>	1378
	registers from task TCB. get floating-point	<code>fppTaskRegsGet()</code>	605
	print summary of each task's TCB.	<code>i()</code>	656
	display task information from TCBS.	<code>taskShow()</code>	1343
	get MIB-II TCP connection table entry.	<code>m2TcpConnEntryGet()</code>	797
	state. set TCP connection to closed	<code>m2TcpConnEntrySet()</code>	798
	all resources used to access TCP group. delete	<code>m2TcpDelete()</code>	798
	routines. TCP information display	<code>tcpShow</code>	313
	debugging information for TCP protocol. display	<code>tcpDebugShow()</code>	1357
	display all statistics for TCP protocol.	<code>tcpstatShow()</code>	1358
	initialize TCP show routines.	<code>tcpShowInit()</code>	1357
	from user data and send it to TCP socket. create zbuf.	<code>zbufSockBufSend()</code>	1502
	receive data in zbuf from TCP socket.	<code>zbufSockRecv()</code>	1505
	send zbuf data to TCP socket.	<code>zbufSockSend()</code>	1507
	initialize MIB-II TCP-group access.	<code>m2TcpInit()</code>	799
	MIB-II TCP-group API for SNMP agents.	<code>m2TcpLib</code>	150
	get MIB-II TCP-group scalar variables.	<code>m2TcpGroupInfoGet()</code>	799
	generic attach routine for TCP/IP network stack.	<code>ipAttach()</code>	714
	generic detach routine for TCP/IP network stack.	<code>ipDetach()</code>	714
	buffers (VxFusion). allocate telegram buffer from pool of	<code>distTBufAlloc()</code>	556
	distributed objects telegram buffer library /	<code>distTBufLib</code>	85
	buffers (VxFusion). return telegram buffer to pool of	<code>distTBufFree()</code>	557
	initialize telnet services.	<code>telnetdInit()</code>	1359
	initialize telnet services.	<code>telnetdStart()</code>	1361
	close active telnet session.	<code>telnetdExit()</code>	1359
	command interpreter for telnet sessions. specify	<code>telnetdParserSet()</code>	1360
	tasks were pre-started by telnetd. report whether.	<code>telnetdStaticTaskInitializationGet()</code>	1362
	(Unimplemented)/ create temporary binary file	<code>tmpfile()</code>	1391
	generate temporary file name (ANSI).	<code>tmpnam()</code>	1391
	create pseudo terminal.	<code>ptyDevCreate()</code>	1043
	destroy pseudo terminal.	<code>ptyDevRemove()</code>	1044
	serial channels. provide terminal device access to	<code>ttyDrv</code>	326
	terminate all RIP processing.	<code>ripShutdown()</code>	1104
	terminate FTP server task.	<code>ftpdDelete()</code>	625
	terminate hash table.	<code>hashTblTerminate()</code>	650
	wait for thread to terminate (POSIX).	<code>pthread_join()</code>	1028
	wake-up list. terminate <code>select()</code>	<code>selWakeupListTerm()</code>	1174
	terminate thread (POSIX).	<code>pthread_exit()</code>	1026
	function (C++). set terminate to user-defined	<code>set_terminate()</code>	1199
	cause abnormal program termination (ANSI).	<code>abort()</code>	403
	call function at program termination (Unimplemented)/	<code>atexit()</code>	421
	bus. test and set location across	<code>sysBusTas()</code>	1301
	C-callable atomic test-and-set primitive	<code>vxTas()</code>	1459
	already loaded into MUX. tests whether device is	<code>muxDevExists()</code>	876

	Keyword	Name	Page
SCSI device. issue	TEST_UNIT_READY command to .....	scsiTestUnitRdy()	1166
write-protect	text segment (VxVMI) .....	vmTextProtect()	1445
transfer file via	TFTP .....	tftpCopy()	1368
Trivial File Transfer Protocol	(TFTP) client library .....	tftpLib	319
send	TFTP message to remote system .....	tftpSend()	1375
set	TFTP server address .....	tftpPeerSet()	1374
	TFTP server daemon task .....	tftpdTask()	1371
initialize	TFTP server task .....	tftpdInit()	1370
initialize	TFTP session .....	tftpInit()	1373
quit	TFTP session .....	tftpQuit()	1375
get	TFTP status information .....	tftpInfoShow()	1372
set	TFTP transfer mode .....	tftpModeSet()	1373
transfer file via	TFTP using stream interface .....	tftpXfer()	1376
get name of	thread attribute object .....	pthread_attr_getname()	1009
set name in	thread attribute object .....	pthread_attr_setname()	1015
set inheritsched attribute in	thread attribute object/ .....	pthread_attr_setinheritsched()	1014
(POSIX). destroy	thread attributes object .....	pthread_attr_destroy()	1007
/of detachstate attribute in	thread attributes object/ .....	pthread_attr_getdetachstate()	1008
/of inheritsched attribute in	thread attributes object/ .....	pthread_attr_getinheritsched()	1008
/of schedparam attribute in	thread attributes object/ .....	pthread_attr_getschedparam()	1009
get schedpolicy attribute from	thread attributes object/ .....	pthread_attr_getschedpolicy()	1010
/of stackaddr attribute from	thread attributes object/ .....	pthread_attr_getstackaddr()	1011
/of stacksize attribute in	thread attributes object/ .....	pthread_attr_getstacksize()	1012
(POSIX). initialize	thread attributes object .....	pthread_attr_init()	1012
set detachstate attribute in	thread attributes object/ .....	pthread_attr_setdetachstate()	1013
set schedparam attribute in	thread attributes object/ .....	pthread_attr_setschedparam()	1015
set schedpolicy attribute in	thread attributes object/ .....	pthread_attr_setschedpolicy()	1016
set stackaddr attribute in	thread attributes object/ .....	pthread_attr_setstackaddr()	1017
set stacksize attribute in	thread attributes object/ .....	pthread_attr_setstacksize()	1018
get contention scope from	thread attributes (POSIX) .....	pthread_attr_getscope()	1011
set contention scope for	thread attributes (POSIX) .....	pthread_attr_setscope()	1017
compare	thread IDs (POSIX) .....	pthread_equal()	1026
perform generic SCSI	thread initialization .....	scsiThreadInit()	1166
POSIX 1003.1c	thread library interfaces .....	pthreadLib	217
cancel execution of	thread (POSIX) .....	pthread_cancel()	1018
create	thread (POSIX) .....	pthread_create()	1025
dynamically detach	thread (POSIX) .....	pthread_detach()	1025
terminate	thread (POSIX) .....	pthread_exit()	1026
of schedparam attribute from	thread (POSIX). get value .....	pthread_getschedparam()	1027
send signal to	thread (POSIX) .....	pthread_kill()	1030
cancellation state for calling	thread (POSIX). set .....	pthread_setcancelstate()	1039
cancellation type for calling	thread (POSIX). set .....	pthread_setcanceltype()	1040
set schedparam attribute for	thread (POSIX). dynamically .....	pthread_setschedparam()	1040
cancellation point in calling	thread (POSIX). create .....	pthread_testcancel()	1043
(POSIX). create	thread specific data key .....	pthread_key_create()	1029
(POSIX). delete	thread specific data key .....	pthread_key_delete()	1029
get	thread specific data (POSIX) .....	pthread_getspecific()	1027
set	thread specific data (POSIX) .....	pthread_setspecific()	1041
send event to	thread state machine .....	scsiMgrThreadEvent()	1148
wait for	thread to terminate (POSIX) .....	pthread_join()	1028

	Keyword	Name	Page
	(POSIX). unblock thread waiting on condition .....	<b>pthread_cond_signal()</b>	1022
library (SCSI-2). SCSI thread-level controller .....		<b>scsiCtrlLib</b>	256
get calling thread's ID (POSIX).....		<b>pthread_self()</b>	1039
change and/or examine calling thread's signal mask (POSIX).....		<b>pthread_sigmask()</b>	1042
initialize POSIX threads support.....		<b>pthreadLibInit()</b>	1007
(POSIX). unblock all threads waiting on condition.....		<b>pthread_cond_broadcast()</b>	1020
initialize TI TMS390 cache library.....		<b>cacheTiTms390LibInit()</b>	473
get value of kernel's tick counter.....		<b>tickGet()</b>	1379
set value of kernel's tick counter.....		<b>tickSet()</b>	1380
clock tick support library.....		<b>tickLib</b>	321
announce clock tick to kernel.....		<b>tickAnnounce()</b>	1379
delay for specified amount of time.....		<b>sleep()</b>	1231
to another one byte at a time. copy one buffer .....		<b>bcopyBytes()</b>	426
to another one long word at a time. copy one buffer .....		<b>bcopyLongs()</b>	426
to another one word at a time. copy one buffer .....		<b>bcopyWords()</b>	427
character one byte at a time. /buffer with specified.....		<b>bfillBytes()</b>	429
network with DHCP at boot time. initialize .....		<b>dhcpcBootBind()</b>	521
determine current calendar time (ANSI).....		<b>time()</b>	1380
time into UTC broken-down time (ANSI). convert calendar.....		<b>gmtime()</b>	641
calendar time into broken-down time (ANSI). convert.....		<b>localtime()</b>	745
broken-down time into calendar time (ANSI). convert.....		<b>mktime()</b>	822
reload value/ get remaining time before expiration and .....		<b>timer_gettime()</b>	1384
ANSI time documentation.....		<b>ansiTime</b>	25
retrieve current time from remote source.....		<b>sntpctimeGet()</b>	1249
(ANSI). convert time in seconds into string.....		<b>ctime()</b>	510
(POSIX). convert time in seconds into string.....		<b>ctime_r()</b>	510
determine processor time in use (ANSI).....		<b>clock()</b>	494
suspend current task until time interval elapses (POSIX).....		<b>nanosleep()</b>	906
(ANSI). convert calendar time into broken-down time.....		<b>localtime_r()</b>	745
(POSIX). convert calendar time into broken-down time.....		<b>gmtime_r()</b>	641
(POSIX). convert calendar time into broken-down time.....		<b>localtime_r_r()</b>	746
(ANSI). convert broken-down time into calendar time.....		<b>mktime_r()</b>	822
(ANSI). convert broken-down time into formatted string.....		<b>strftime()</b>	1275
convert broken-down time into string (ANSI).....		<b>asctime()</b>	416
convert broken-down time into string (POSIX).....		<b>asctime_r()</b>	416
(ANSI). convert calendar time into UTC broken-down time .....		<b>gmtime_r_r()</b>	641
get current time of clock (POSIX).....		<b>clock_gettime()</b>	495
update time on file.....		<b>utime()</b>	1427
set clock to specified time (POSIX).....		<b>clock_settime()</b>	496
calendar time into broken-down time (POSIX). convert .....		<b>gmtime_r_r_r()</b>	641
calendar time into broken-down time (POSIX). convert .....		<b>localtime_r_r_r()</b>	746
library. Simple Network Time Protocol (SNTP) client.....		<b>sntpLib</b>	291
library. Simple Network Time Protocol (SNTP) server .....		<b>sntpsLib</b>	292
function or group of/ time repeated executions of .....		<b>timexN()</b>	1388
function or functions. time single execution of.....		<b>timex()</b>	1385
get current time slice (POSIX).....		<b>sched_rr_get_interval()</b>	1134
arm timer (POSIX). set time until next expiration and .....		<b>timer_settime()</b>	1384
list of function calls to be timed. clear.....		<b>timexClear()</b>	1386
specify functions to be timed.....		<b>timexFunc()</b>	1386
list of function calls to be timed. display .....		<b>timexShow()</b>	1390

	Keyword	Name	Page
called after watchdog	timeout. ....	<b>rdiscTimerEvent()</b>	1063
set WTX	timeout. ....	<b>symSyncTimeoutSet()</b>	1295
for condition variable with	timeout (POSIX). wait .....	<b>pthread_cond_timedwait()</b>	1022
cancel	timer. ....	<b>timer_cancel()</b>	1381
create watchdog	timer. ....	<b>wdCreate()</b>	1465
delete watchdog	timer. ....	<b>wdDelete()</b>	1465
start watchdog	timer. ....	<b>wdStart()</b>	1467
(POSIX). return	timer expiration overrun .....	<b>timer_getoverrun()</b>	1383
display synopsis of execution	timer facilities. ....	<b>timexHelp()</b>	1387
execution	timer facilities. ....	<b>timexLib</b>	323
include execution	timer library. ....	<b>timexInit()</b>	1388
watchdog	timer library. ....	<b>wdLib</b>	349
	timer library (POSIX). ....	<b>timerLib</b>	322
	timer library (WindView). ....	<b>wvTmrLib</b>	357
remove previously created	timer (POSIX). ....	<b>timer_delete()</b>	1383
until next expiration and arm	timer (POSIX). set time.....	<b>timer_settime()</b>	1384
connect user routine to	timer signal. ....	<b>timer_connect()</b>	1381
for timing base/ allocate	timer using specified clock .....	<b>timer_create()</b>	1382
register timestamp	timer (WindView). ....	<b>wvTmrRegister()</b>	1485
retrieve current lease	timers. ....	<b>dhcpcTimerGet()</b>	537
display current lease	timers. ....	<b>dhcpcTimersShow()</b>	537
between two calendar	times (ANSI). /difference .....	<b>difftime()</b>	542
get lower half of 64Bit TSC	(Timestamp Counter). ....	<b>pentiumTscGet32()</b>	974
get 64Bit TSC	(Timestamp Counter). ....	<b>pentiumTscGet64()</b>	974
reset TSC	(Timestamp Counter). ....	<b>pentiumTscReset()</b>	974
register	timestamp timer (WindView). ....	<b>wvTmrRegister()</b>	1485
functions to be called after	timing, specify .....	<b>timexPost()</b>	1389
to be called prior to	timing, specify functions.....	<b>timexPre()</b>	1390
using specified clock for	timing base (POSIX). /timer.....	<b>timer_create()</b>	1382
Buffers). flush	TLBs (Translation Lookaside.....	<b>pentiumTlbFlush()</b>	973
initialize TI	TMS390 cache library.....	<b>cacheTiTms390LibInit()</b>	473
adds device to list polled by	tMuxPollTask.....	<b>muxPollDevAdd()</b>	887
device from list polled by	tMuxPollTask. removes.....	<b>muxPollDevDel()</b>	887
device is on list polled by	tMuxPollTask. reports whether.....	<b>muxPollDevStat()</b>	888
get priority of	tMuxPollTask.....	<b>muxTaskPriorityGet()</b>	894
reset priority of	tMuxPollTask.....	<b>muxTaskPrioritySet()</b>	895
devices to/ shuts down	tMuxPollTask and returns .....	<b>muxPollEnd()</b>	888
break down string into	tokens (ANSI). ....	<b>strtok()</b>	1282
break down string into	tokens (reentrant) (POSIX). ....	<b>strtok_r()</b>	1283
task CPU utilization	tool package. initialize.....	<b>spyLibInit()</b>	1259
Library. MUX	Toolkit Network Interface .....	<b>muxTkLib</b>	188
interface. send packet out on	Toolkit or END network .....	<b>muxTkSend()</b>	902
post user event string to host	tools.....	<b>wdbUserEvtPost()</b>	1463
library.	Toshiba Tx49 cache management .....	<b>cacheTx49Lib</b>	54
display stack	trace of task. ....	<b>tt()</b>	1404
memory.	transfer blocks to or from.....	<b>cbioBlkRW()</b>	477
memory.	transfer bytes to or from .....	<b>cbioBytesRW()</b>	477
reset network devices and	transfer control to boot ROMs. ....	<b>reboot()</b>	1066
monitor.	transfer control to ROM.....	<b>sysToMonitor()</b>	1317
	transfer file via TFTP.....	<b>tftpCopy()</b>	1368

	Keyword	Name	Page
	stream interface.	transfer file via TFTP using.....	<b>tftpXfer()</b> 1376
	(WindView).	transfer log header to host .....	<b>wvLogHeaderUpload()</b> 1474
	set TFTP	transfer mode. ....	<b>tftpModeSet()</b> 1373
	or continue negotiating	transfer parameters. initiate.....	<b>scsiSyncXferNegotiate()</b> 1162
	library. File	Transfer Protocol (FTP).....	<b>ftpLib</b> 115
	server. File	Transfer Protocol (FTP).....	<b>ftpdLib</b> 113
	library. Trivial File	Transfer Protocol server.....	<b>tftpdLib</b> 318
	client library. Trivial File	Transfer Protocol (TFTP) .....	<b>tftpLib</b> 319
	to block (sector to sector)	transfer routine. block.....	<b>cbioBlkCopy()</b> 476
	initiate	transfer via FTP.....	<b>ftpXfer()</b> 631
	of s2 into s1 (ANSI).	transform up to n characters.....	<b>strxfrm()</b> 1287
	set applette to stop FTP	transient host responses. ....	<b>ftpTransientFatalInstall()</b> 631
	Buffers). flush TLBs	(Translation Lookaside .....	<b>pentiumTlbFlush()</b> 973
	SimSolaris,/ set CPU vector	(trap) (68K, x86, MIPS, SH, .....	<b>intVecSet()</b> 699
	MIPS, ARM,/ get vector	(trap) base address (68K, x86, .....	<b>intVecBaseGet()</b> 696
	MIPS, ARM,/ set vector	(trap) base address (68K, x86, .....	<b>intVecBaseSet()</b> 697
	change	trap-to-monitor character. ....	<b>tyMonitorTrapSet()</b> 1411
		traverse IP routing table. ....	<b>routeTableWalk()</b> 1123
	enable	trigger. ....	<b>trgEnable()</b> 1396
	library.	trigger events control .....	<b>trgLib</b> 324
	delete	trigger from trigger list.....	<b>trgDelete()</b> 1395
	show	trigger information.....	<b>trgShow()</b> 1399
	add new trigger to	trigger list. ....	<b>trgAdd()</b> 1393
	delete trigger from	trigger list. ....	<b>trgDelete()</b> 1395
	turn	trigger off.....	<b>trgDisable()</b> 1396
	initialize	trigger show facility. ....	<b>trgShowInit()</b> 1399
		trigger show routine. ....	<b>trgShow</b> 325
	add new	trigger to trigger list.....	<b>trgAdd()</b> 1393
		trigger user-defined event.....	<b>trgEvent()</b> 1397
	queue. reset	trigger work queue task and .....	<b>trgWorkQReset()</b> 1400
	initialize	triggering library.....	<b>trgLibInit()</b> 1397
	set	triggering off. ....	<b>trgOff()</b> 1398
	set	triggering on. ....	<b>trgOn()</b> 1398
	chains two	triggers. ....	<b>trgChainSet()</b> 1395
	server library.	Trivial File Transfer Protocol.....	<b>tftpdLib</b> 318
	(TFTP) client library.	Trivial File Transfer Protocol.....	<b>tftpLib</b> 319
	flash device for use with	TrueFFS. format .....	<b>tffsDevFormat()</b> 1364
	for use with dosFs. create	TrueFFS block device suitable .....	<b>tffsDevCreate()</b> 1363
	VxWorks.	TrueFFS configuration file for.....	<b>tffsConfig</b> 315
		TrueFFS interface for VxWorks.....	<b>tffsDrv</b> 316
	initialize	TrueFFS system.....	<b>tffsDrv()</b> 1365
	set	TrueFFS volume options. ....	<b>tffsDevOptionsSet()</b> 1365
		truncate file (POSIX). ....	<b>ftruncate()</b> 633
		truncate to integer. ....	<b>trunc()</b> 1400
		truncate to integer. ....	<b>trunc()</b> 1401
	POSIX file	truncation. ....	<b>ftruncate</b> 117
	get lower half of 64Bit	TSC (Timestamp Counter). ....	<b>pentiumTscGet32()</b> 974
	get 64Bit	TSC (Timestamp Counter). ....	<b>pentiumTscGet64()</b> 974
	reset	TSC (Timestamp Counter). ....	<b>pentiumTscReset()</b> 974
	host connection library using	TSFS. target.....	<b>wvTsfsUploadPathLib</b> 358

	Keyword	Name	Page
open upload path to host using	TSFS socket (Windview). .....	<b>tsfsUploadPathCreate()</b>	1402
write to	TSFS upload path (Windview). .....	<b>tsfsUploadPathWrite()</b>	1403
(Windview). close	TSFS-socket upload path .....	<b>tsfsUploadPathClose()</b>	1402
do task-level read for	tty device. ....	<b>tyRead()</b>	1411
do task-level write for	tty device. ....	<b>tyWrite()</b>	1412
whether underlying driver is	tty device. return .....	<b>isatty()</b>	720
initialize	tty device descriptor. ....	<b>tyDevInit()</b>	1408
remove	tty device descriptor. ....	<b>tyDevRemove()</b>	1408
initialize	tty driver. ....	<b>ttyDrv()</b>	1405
modify	tty driver support library. ....	<b>tyLib</b>	326
set priority and stack size of	tunable disk cache parameters. ....	<b>dcacheDevTune()</b>	517
initialize	tWVUpload task (WindView). ....	<b>wvUploadTaskConfig()</b>	1488
Toshiba	Tx49 cache library. ....	<b>cacheTx49LibInit()</b>	475
Toshiba	Tx49 cache management library. ....	<b>cacheTx49Lib</b>	54
selector/ get CPU vector, gate	type(int/trap), and gate .....	<b>intVecGet2()</b>	699
(x86). set CPU vector, gate	type(int/trap), and selector .....	<b>intVecSet2()</b>	703
all resources used to access	UDP group. delete .....	<b>m2UdpDelete()</b>	800
routines.	UDP information display .....	<b>udpShow</b>	332
get UDP MIB-II entry from	UDP list of listeners. ....	<b>m2UdpTblEntryGet()</b>	801
list of listeners. get	UDP MIB-II entry from UDP. ....	<b>m2UdpTblEntryGet()</b>	801
display statistics for	UDP protocol. ....	<b>udpstatShow()</b>	1413
initialize	UDP show routines. ....	<b>udpShowInit()</b>	1413
user message and send it to	UDP socket. create zbuf from. ....	<b>zbufSockBufSendto()</b>	1503
receive message in zbuf from	UDP socket. ....	<b>zbufSockRecvfrom()</b>	1506
send zbuf message to	UDP socket. ....	<b>zbufSockSendto()</b>	1508
initialize MIB-II	UDP-group access. ....	<b>m2UdpInit()</b>	801
MIB-II	UDP-group API for SNMP agents. ....	<b>m2UdpLib</b>	152
get MIB-II	UDP-group scalar variables. ....	<b>m2UdpGroupInfoGet()</b>	800
condition (POSIX).	unlock all threads waiting on .....	<b>pthread_cond_broadcast()</b>	1020
semaphore.	unlock every task pended on .....	<b>semFlush()</b>	1181
condition (POSIX).	unlock thread waiting on .....	<b>pthread_cond_signal()</b>	1022
string to command processor	(Unimplemented) (ANSI). pass .....	<b>system()</b>	1317
create temporary binary file	(Unimplemented) (ANSI). ....	<b>tmpfile()</b>	1391
char's to multibyte char's	(Unimplemented) (ANSI). /wide .....	<b>wcstombs()</b>	1461
/to multibyte character	(Unimplemented) (ANSI). ....	<b>wctomb()</b>	1461
/at program termination	(Unimplemented) (ANSI). ....	<b>atexit()</b>	421
/length of multibyte character	(Unimplemented) (ANSI). ....	<b>mblen()</b>	804
/char's to wide char's	(Unimplemented) (ANSI). ....	<b>mbstowcs()</b>	804
/character to wide character	(Unimplemented) (ANSI). ....	<b>mbtowc()</b>	805
(ARM). set	uninitialized vector handler .....	<b>intUninitVecSet()</b>	695
dosFs disk on top of	UNIX. initialize .....	<b>unixDiskInit()</b>	1415
parameters. get NFS	UNIX authentication .....	<b>nfsAuthUnixGet()</b>	932
parameters. modify NFS	UNIX authentication .....	<b>nfsAuthUnixPrompt()</b>	933
parameters. set NFS	UNIX authentication .....	<b>nfsAuthUnixSet()</b>	933
parameters. display NFS	UNIX authentication .....	<b>nfsAuthUnixShow()</b>	934
set ID number of NFS	UNIX authentication/ .....	<b>nfsIdSet()</b>	941
create	UNIX BSD 4.3 select library. ....	<b>selectLib</b>	261
install	UNIX disk device. ....	<b>unixDiskDevCreate()</b>	1414
(VxSim). pass-through (to	UNIX disk driver. ....	<b>unixDrv()</b>	1416
(VxSim). pass-through (to	UNIX) file system library .....	<b>passFsLib</b>	200

	Keyword	Name	Page
	UNIX tar compatible library.....	<b>tarLib</b>	306
for Solaris and VxSim for/ /module ID information (use specifying file name or/ specifying group number. specifying module ID. specifying name and path. object module	UNIX-file disk driver (VxSim..... <b>unld()</b> to reclaim space).....	<b>unixDrv</b> <b>moduleDelete()</b>	332 830
	unload object module by.....	<b>unld()</b>	1416
	unload object module by.....	<b>unldByGroup()</b>	1417
	unload object module by.....	<b>unldByModuleId()</b>	1417
	unload object module by.....	<b>unldByNameAndPath()</b>	1418
	unloading library.....	<b>unldLib</b>	334
	unloads device from MUX.....	<b>muxDevUnload()</b>	879
specified cache.	unlock all or part of.....	<b>cacheUnlock()</b>	475
process (POSIX).	unlock all pages used by.....	<b>munlockall()</b>	869
(POSIX).	unlock (give) semaphore.....	<b>sem_post()</b>	1193
	unlock mutex (POSIX).....	<b>pthread_mutex_unlock()</b>	1034
(POSIX).	unlock specified pages.....	<b>munlock()</b>	869
	unmount NFS device.....	<b>nfsUnmount()</b>	943
destroy	unnamed semaphore (POSIX).....	<b>sem_destroy()</b>	1189
initialize	unnamed semaphore (POSIX).....	<b>sem_init()</b>	1191
configure interface to be	unnumbered.....	<b>ifUnnumberedSet()</b>	672
	unregister proxy client.....	<b>proxyUnreg()</b>	1005
make calling task	unsafe from deletion.....	<b>taskUnsafe()</b>	1352
convert string to	unsigned long integer (ANSI).....	<b>strtoul()</b>	1285
non-counter object.	update contents of interface.....	<b>m2IfVariableUpdate()</b>	785
register (MIPS).	update contents of status.....	<b>intSRSet()</b>	694
interface. remove	update filter from RIP.....	<b>ripSendHookDelete()</b>	1104
interface. add	update filter to RIP.....	<b>ripSendHookAdd()</b>	1103
sub-layers.	update relationship between.....	<b>m2IfStackTblUpdate()</b>	782
install interface counter	update routine.....	<b>m2IfCtrUpdateRtnInstall()</b>	775
install interface variable	update routine.....	<b>m2IfVarUpdateRtnInstall()</b>	786
	update time on file.....	<b>utime()</b>	1427
enable last access date	updating for this volume.....	<b>dosFsLastAccessDateEnable()</b>	560
(WindView). start	upload of events to host.....	<b>wvUploadStart()</b>	1486
(WindView). stop	upload of events to host.....	<b>wvUploadStop()</b>	1487
socket	upload path library.....	<b>wvSockUploadPathLib</b>	357
socket (Windview). establish	upload path to host using.....	<b>sockUploadPathCreate()</b>	1254
socket (Windview). open	upload path to host using TSFS.....	<b>tsfsUploadPathCreate()</b>	1402
close socket	upload path (Windview).....	<b>sockUploadPathClose()</b>	1253
write to socket	upload path (Windview).....	<b>sockUploadPathWrite()</b>	1255
close TSFS-socket	upload path (Windview).....	<b>tsfsUploadPathClose()</b>	1402
write to TSFS	upload path (Windview).....	<b>tsfsUploadPathWrite()</b>	1403
events (WindView).	upload preserved task name.....	<b>wvTaskNamesUpload()</b>	1485
convert lower-case letter to	upper-case equivalent (ANSI).....	<b>toupper()</b>	1392
test whether character is	upper-case letter (ANSI).....	<b>isupper()</b>	724
lower-case equivalent/ convert	upper-case letter to.....	<b>tolower()</b>	1392
delete module ID information	(use <b>unld()</b> to reclaim space).....	<b>moduleDelete()</b>	830
socket. create zbuf from	user data and send it to TCP.....	<b>zbufSockBufSend()</b>	1502
login prompt and validate	user entry. display.....	<b>loginPrompt()</b>	753
delete	user entry from login table.....	<b>loginUserDelete()</b>	755
include WDB	user event library.....	<b>wdbUserEvtLibInit()</b>	1463
WDB	user event library.....	<b>wdbUserEvtLib</b>	348
tools. post	user event string to host.....	<b>wdbUserEvtPost()</b>	1463

	Keyword	Name	Page
library. file system	user interface subroutine .....	<b>usrFsLib</b>	339
library.	user interface subroutine .....	<b>usrLib</b>	341
display	user login table. ....	<b>loginUserShow()</b>	756
library.	user login/password subroutine.....	<b>loginLib</b>	134
UDP socket. create zbuf from	user message and send it to.....	<b>zbufSockBufSendto()</b>	1503
get current	user name and password. ....	<b>remCurIdGet()</b>	1070
set remote	user name and password. ....	<b>remCurIdSet()</b>	1070
set remote	user name and password. ....	<b>iam()</b>	657
login table. verify	user name and password in .....	<b>loginUserVerify()</b>	756
connect	user routine to timer signal. ....	<b>timer_connect()</b>	1381
add	user to login table.....	<b>loginUserAdd()</b>	754
trigger	user-defined event. ....	<b>trgEvent()</b>	1397
log	user-defined event (WindView).....	<b>wvEvent()</b>	1469
set new_handler to	user-defined function (C++). ....	<b>set_new_handler()</b>	1199
set terminate to	user-defined function (C++). ....	<b>set_terminate()</b>	1199
interrupt routine.	user-defined system clock .....	<b>usrClock()</b>	1420
configuration library.	user-defined system.....	<b>usrConfig</b>	336
initialization routine.	user-defined system.....	<b>usrInit()</b>	1424
convert calendar time into	UTC broken-down time (ANSI). ....	<b>gmtime()</b>	641
initialize task CPU	utilization tool package.....	<b>spyLibInit()</b>	1259
/va_list object for use by	<b>va_arg()</b> and <b>va_end()</b> . ....	<b>va_start()</b>	1429
object for use by <b>va_arg()</b> and	<b>va_end()</b> . initialize <b>va_list</b> .....	<b>va_start()</b>	1429
driver-specific value.	validate open fd and return.....	<b>iosFdValue()</b>	711
display login prompt and	validate user entry.....	<b>loginPrompt()</b>	753
return from routine using	<b>va_list</b> object. /normal .....	<b>va_end()</b>	1428
<b>va_arg()</b> and/ initialize	<b>va_list</b> object for use by .....	<b>va_start()</b>	1429
SimSolaris,/ get interrupt	vector (68K, x86, MIPS, SH, .....	<b>intVecGet()</b>	698
and gate selector/ get CPU	vector, gate type(int/trap), .....	<b>intVecGet2()</b>	699
and selector (x86). set CPU	vector, gate type(int/trap), .....	<b>intVecSet2()</b>	703
set uninitialized	vector handler (ARM). ....	<b>intUninitVecSet()</b>	695
routine to critical exception	vector (PowerPC 403). /C.....	<b>excCrtConnect()</b>	578
routine to critical interrupt	vector (PowerPC 403). /C.....	<b>excIntCrtConnect()</b>	581
connect C routine to exception	vector (PowerPC). ....	<b>excConnect()</b>	577
/to asynchronous exception	vector (PowerPC, ARM). ....	<b>excIntConnect()</b>	580
get CPU exception	vector (PowerPC, ARM). ....	<b>excVecGet()</b>	582
set CPU exception	vector (PowerPC, ARM). ....	<b>excVecSet()</b>	584
write-protect exception	vector table (68K, x86, ARM,/ .....	<b>intVecTableWriteProtect()</b>	704
SH, SimSolaris,/ set CPU	vector (trap) (68K, x86, MIPS, .....	<b>intVecSet()</b>	699
(68K, x86, MIPS, ARM,/ get	vector (trap) base address.....	<b>intVecBaseGet()</b>	696
(68K, x86, MIPS, ARM,/ set	vector (trap) base address.....	<b>intVecBaseSet()</b>	697
initialize exception/interrupt	vectors.....	<b>excVecInit()</b>	582
display all IP routes	(verbose information). ....	<b>mRouteShow()</b>	849
return BSP	version and revision number. ....	<b>sysBspRev()</b>	1299
print VxWorks	version information. ....	<b>version()</b>	1430
kernel heap	version of <b>netPoolInit()</b> . ....	<b>netPoolKheapInit()</b>	927
translate physical address to	virtual address (ARM). ....	<b>mmuPhysToVirt()</b>	823
translate	virtual address for cacheLib.....	<b>cacheTITms390VirtToPhys()</b>	474
translate	virtual address for drivers. ....	<b>cacheDrvVirtToPhys()</b>	455
address (ARM). translate	virtual address to physical .....	<b>mmuVirtToPhys()</b>	826
address (VxVMI). translate	virtual address to physical .....	<b>vmTranslate()</b>	1445

	Keyword	Name	Page
virtual space in shared global	virtual mem (VxVMI). /pages to.....	<b>vmGlobalMap()</b>	1438
virtual/ map physical pages to	virtual space in shared global.....	<b>vmGlobalMap()</b>	1438
map physical space into	virtual space (VxVMI). .....	<b>vmMap()</b>	1440
change state of block of	virtual memory. ....	<b>vmBaseStateSet()</b>	1433
(VxVMI). create new	virtual memory context.....	<b>vmContextCreate()</b>	1434
(VxVMI). delete	virtual memory context.....	<b>vmContextDelete()</b>	1435
(VxVMI). get current	virtual memory context.....	<b>vmCurrentGet()</b>	1436
(VxVMI). set current	virtual memory context.....	<b>vmCurrentSet()</b>	1436
(VxVMI). get global	virtual memory information.....	<b>vmGlobalInfoGet()</b>	1437
(VxVMI). include	virtual memory show facility .....	<b>vmShowInit()</b>	1442
(VxVMI).	virtual memory show routines.....	<b>vmShow</b>	346
initialize base	virtual memory support.....	<b>vmBaseLibInit()</b>	1432
library. base	virtual memory support.....	<b>vmBaseLib</b>	343
architecture-independent	virtual memory support library/.....	<b>vmLib</b>	343
(VxVMI). initialize	virtual memory support module .....	<b>vmLibInit()</b>	1440
enable or disable	virtual memory (VxVMI). .....	<b>vmEnable()</b>	1437
get state of page of	virtual memory (VxVMI). .....	<b>vmStateGet()</b>	1443
change state of block of	virtual memory (VxVMI). .....	<b>vmStateSet()</b>	1444
modify mode of raw device	volume. ....	<b>rawFsModeChange()</b>	1055
disable raw device	volume. ....	<b>rawFsVolUnmount()</b>	1056
modify mode of rt11Fs	volume. ....	<b>rt11FsModeChange()</b>	1128
disable tape device	volume. ....	<b>tapeFsVolUnmount()</b>	1322
access date updating for this	volume. enable last.....	<b>dosFsLastAccessDateEnable()</b>	560
format MS-DOS compatible	volume. ....	<b>dosFsVolFormat()</b>	562
set case sensitivity of	volume. ....	<b>dosSetVolCaseSens()</b>	563
display dosFs	volume configuration data.....	<b>dosFsShow()</b>	561
information. show	volume configuration .....	<b>cdromFsVolConfigShow()</b>	489
convert device name into DOS	volume descriptor pointer.....	<b>dosFsVolDescGet()</b>	561
block device with raw	volume functions. associate.....	<b>rawFsDevInit()</b>	1054
sequential device with tape	volume functions. associate .....	<b>tapeFsDevInit()</b>	1320
make	volume integrity checking. ....	<b>dosFsChkDsk()</b>	558
prepare to use raw	volume library.....	<b>rawFsInit()</b>	1055
initialize tape	volume library.....	<b>tapeFsInit()</b>	1321
set TrueFFS	volume options. ....	<b>tffsDevOptionsSet()</b>	1365
message queue group library	(VxFusion). distributed .....	<b>msgQDistGrpLib</b>	180
queue group show routines	(VxFusion). /message.....	<b>msgQDistGrpShow</b>	181
objects message queue library	(VxFusion). distributed .....	<b>msgQDistLib</b>	181
message queue show routines	(VxFusion). distributed .....	<b>msgQDistShow</b>	182
objects control function	(VxFusion). /distributed .....	<b>distCtl()</b>	544
installed interface adapter	(VxFusion). /information about .....	<b>distIfShow()</b>	548
and bootstrap current node	(VxFusion). initialize.....	<b>distInit()</b>	549
to distributed name database	(VxFusion). add entry .....	<b>distNameAdd()</b>	551
name database filtered by type	(VxFusion). /distributed .....	<b>distNameFilterShow()</b>	552
by name in local database	(VxFusion). find object.....	<b>distNameFind()</b>	553
of object by value and type	(VxFusion). look up name.....	<b>distNameFindByValueAndType()</b>	554
from distributed name database	(VxFusion). remove entry .....	<b>distNameRemove()</b>	555
distributed name database	(VxFusion). display entire.....	<b>distNameShow()</b>	555
buffer from pool of buffers	(VxFusion). allocate telegram .....	<b>distTBufAlloc()</b>	556
buffer to pool of buffers	(VxFusion). return telegram .....	<b>distTBufFree()</b>	557
adapter show routines	(VxFusion). /objects interface .....	<b>distIfShow</b>	83

	Keyword	Name	Page
and control library	(VxFusion). /initialization.....	<b>distLib</b>	83
name database library	(VxFusion). distributed .....	<b>distNameLib</b>	84
distributed message queue	(VxFusion). create .....	<b>msgQDistCreate()</b>	851
message queue to group	(VxFusion). add distributed .....	<b>msgQDistGrpAdd()</b>	853
message queue from group	(VxFusion). /distributed.....	<b>msgQDistGrpDelete()</b>	854
all or one group with members	(VxFusion). display.....	<b>msgQDistGrpShow()</b>	854
in distributed message queue	(VxFusion). /of messages .....	<b>msgQDistNumMsgs()</b>	855
from distributed message queue	(VxFusion). receive message .....	<b>msgQDistReceive()</b>	856
to distributed message queue	(VxFusion). send message .....	<b>msgQDistSend()</b>	857
message queue show package	(VxFusion). /distributed.....	<b>msgQDistShowInit()</b>	859
name database show routines	(VxFusion). distributed .....	<b>distNameShow</b>	85
telegram buffer library	(VxFusion). /objects .....	<b>distTBufLib</b>	85
architecture-specific part of	<b>vxMemProbe()</b> .....	<b>vxMemArchProbe()</b>	1454
shared memory binary semaphore	(VxMP). create and initialize .....	<b>semBSmCreate()</b>	1175
memory counting semaphore	(VxMP). /and initialize shared .....	<b>semCSmCreate()</b>	1178
shared memory system partition	(VxMP). add memory to .....	<b>smMemAddToPool()</b>	1231
shared memory system partition	(VxMP). /memory for array from .....	<b>smMemCallc()</b>	1232
shared memory system partition	(VxMP). /largest free block in.....	<b>smMemFindMax()</b>	1233
partition block of memory	(VxMP). /shared memory system.....	<b>smMemFree()</b>	1233
shared memory system partition	(VxMP). /block of memory from .....	<b>smMemMalloc()</b>	1234
shared memory system partition	(VxMP). set debug options for .....	<b>smMemOptionsSet()</b>	1234
shared memory system partition	(VxMP). /block of memory from .....	<b>smMemRealloc()</b>	1235
blocks and statistics	(VxMP). /system partition .....	<b>smMemShow()</b>	1236
to shared memory name database	(VxMP). add name.....	<b>smNameAdd()</b>	1237
shared memory object by name	(VxMP). look up .....	<b>smNameFind()</b>	1238
shared memory object by value	(VxMP). look up .....	<b>smNameFindByValue()</b>	1239
memory objects name database	(VxMP). /object from shared .....	<b>smNameRemove()</b>	1240
memory objects name database	(VxMP). /contents of shared.....	<b>smNameShow()</b>	1240
shared memory objects facility	(VxMP). attach calling CPU to .....	<b>smObjAttach()</b>	1242
address to local address	(VxMP). convert global .....	<b>smObjGlobalToLocal()</b>	1243
memory objects descriptor	(VxMP). initialize shared .....	<b>smObjInit()</b>	1244
shared memory objects facility	(VxMP). install.....	<b>smObjLibInit()</b>	1245
address to global address	(VxMP). convert local.....	<b>smObjLocalToGlobal()</b>	1245
shared memory objects facility	(VxMP). initialize .....	<b>smObjSetup()</b>	1246
of shared memory objects	(VxMP). /current status.....	<b>smObjShow()</b>	1247
attempts to take spin-lock	(VxMP). /logging of failed .....	<b>smObjTimeoutLogEnable()</b>	1248
memory message queue library	(VxMP). shared.....	<b>msgQSmLib</b>	185
memory semaphore library	(VxMP). shared.....	<b>semSmLib</b>	274
memory management library	(VxMP). shared.....	<b>smMemLib</b>	281
management show routines	(VxMP). shared memory.....	<b>smMemShow</b>	284
objects name database library	(VxMP). shared memory.....	<b>smNameLib</b>	284
name database show routines	(VxMP). shared memory objects.....	<b>smNameShow</b>	286
shared memory objects library	(VxMP). .....	<b>smObjLib</b>	288
memory objects show routines	(VxMP). shared.....	<b>smObjShow</b>	291
create shared memory partition	(VxMP). .....	<b>memPartSmCreate()</b>	818
shared memory message queue	(VxMP). create and initialize .....	<b>msgQSmCreate()</b>	868
(to UNIX) file system library	(VxSim). pass-through .....	<b>passFsLib</b>	200
driver (VxSim for Solaris and	VxSim for HP). UNIX-file disk.....	<b>unixDrv</b>	332
for/ UNIX-file disk driver	(VxSim for Solaris and VxSim.....	<b>unixDrv</b>	332
new virtual memory context	(VxVMI). create .....	<b>vmContextCreate()</b>	1434

	Keyword	Name	Page
delete virtual memory context	(VxVMI). .....	<b>vmContextDelete()</b>	1435
translation table for context	(VxVMI). display .....	<b>vmContextShow()</b>	1435
current virtual memory context	(VxVMI). get .....	<b>vmCurrentGet()</b>	1436
current virtual memory context	(VxVMI). set .....	<b>vmCurrentSet()</b>	1436
or disable virtual memory	(VxVMI). enable.....	<b>vmEnable()</b>	1437
virtual memory information	(VxVMI). get global.....	<b>vmGlobalInfoGet()</b>	1437
in shared global virtual mem	(VxVMI). /to virtual space .....	<b>vmGlobalMap()</b>	1438
initialize global mapping	(VxVMI). .....	<b>vmGlobalMapInit()</b>	1439
virtual memory support module	(VxVMI). initialize.....	<b>vmLibInit()</b>	1440
space into virtual space	(VxVMI). map physical .....	<b>vmMap()</b>	1440
/page block size	(VxVMI). .....	<b>vmPageBlockSizeGet()</b>	1441
return page size	(VxVMI). .....	<b>vmPageSizeGet()</b>	1442
virtual memory show facility	(VxVMI). include.....	<b>vmShowInit()</b>	1442
of page of virtual memory	(VxVMI). get state .....	<b>vmStateGet()</b>	1443
of block of virtual memory	(VxVMI). change state .....	<b>vmStateSet()</b>	1444
write-protect text segment	(VxVMI). .....	<b>vmTextProtect()</b>	1445
address to physical address	(VxVMI). translate virtual.....	<b>vmTranslate()</b>	1445
/virtual memory support library	(VxVMI). .....	<b>vmLib</b>	343
virtual memory show routines	(VxVMI). .....	<b>vmShow</b>	346
TrueFFS configuration file for	VxWorks. ....	<b>tffsConfig</b>	315
TrueFFS interface for	VxWorks. ....	<b>tffsDrv</b>	316
channel. create	VxWorks device for serial .....	<b>ttyDevCreate()</b>	1405
	VxWorks events library.....	<b>eventLib</b>	104
message queues.	VxWorks events support for.....	<b>msgQEvLib</b>	183
semaphores.	VxWorks events support for.....	<b>semEvLib</b>	265
instrument	VxWorks Events (WindView). .....	<b>wvEventInst()</b>	1469
RIP for SNMP Agent.	VxWorks interface routines to .....	<b>m2RipLib</b>	147
memory network (backplane)/	VxWorks interface to shared.....	<b>smNetLib</b>	287
	VxWorks kernel library.....	<b>kernelLib</b>	129
print	VxWorks logo.....	<b>printLogo()</b>	1001
	VxWorks remote login daemon.....	<b>rlogind()</b>	1105
log out of	VxWorks system. ....	<b>logout()</b>	758
print	VxWorks version information. ....	<b>version()</b>	1430
wake-up list.	wake up all tasks in <b>select()</b> .....	<b>selWakeupAll()</b>	1172
<b>select()</b> .	wake up task pending in .....	<b>selWakeup()</b>	1172
get type of <b>select()</b>	wake-up node. ....	<b>selWakeupType()</b>	1174
wake-up list. add	wake-up node to <b>select()</b> .....	<b>selNodeAdd()</b>	1171
add wake-up node to <b>select()</b>	wake-up list.....	<b>selNodeAdd()</b>	1171
and delete node from <b>select()</b>	wake-up list. find.....	<b>selNodeDelete()</b>	1171
wake up all tasks in <b>select()</b>	wake-up list.....	<b>selWakeupAll()</b>	1172
initialize <b>select()</b>	wake-up list.....	<b>selWakeupListInit()</b>	1173
number of nodes in <b>select()</b>	wake-up list. get .....	<b>selWakeupListLen()</b>	1173
terminate <b>select()</b>	wake-up list.....	<b>selWakeupListTerm()</b>	1174
cancel currently counting	watchdog. ....	<b>wdCancel()</b>	1464
show information about	watchdog. ....	<b>wdShow()</b>	1466
initialize	watchdog show facility.....	<b>wdShowInit()</b>	1466
	watchdog show routines.....	<b>wdShow</b>	350
called after	watchdog timeout. ....	<b>rdiscTimerEvent()</b>	1063
create	watchdog timer.....	<b>wdCreate()</b>	1465
delete	watchdog timer.....	<b>wdDelete()</b>	1465

	Keyword	Name	Page
	start	watchdog timer.....	<b>wdStart()</b> 1467
		watchdog timer library.....	<b>wdLib</b> 349
on SCSI bus (Western Digital	WD33C93 only). /RST line .....	<b>sysScsiBusReset()</b>	1312
library.	WDB agent context management .....	<b>wdbLib</b>	348
include	WDB user event library.....	<b>wdbUserEvtLibInit()</b>	1463
	WDB user event library.....	<b>wdbUserEvtLib</b>	348
report whether tasks	were pre-started by telnetd. <b>telnetdStaticTaskInitializationGet()</b>		1362
assert RST line on SCSI bus	(Western Digital WD33C93/ .....	<b>sysScsiBusReset()</b>	1312
test whether character is	white-space character (ANSI). .....	<b>isspace()</b>	724
character/ convert	wide character to multibyte .....	<b>wctomb()</b>	1461
convert multibyte character to	wide character (Unimplemented)/ .....	<b>mbtowc()</b>	805
char's/ convert series of	wide char's to multibyte.....	<b>wcstombs()</b>	1461
/series of multibyte char's to	wide char's (Unimplemented)/.....	<b>mbstowcs()</b>	804
or continue negotiating	wide parameters. initiate .....	<b>scsiWideXferNegotiate()</b>	1167
copy hierarchy of files with	wildcards.....	<b>xcopy()</b>	1490
delete hierarchy of files with	wildcards.....	<b>xdelete()</b>	1490
library. pass-through (to	Windows NT) file system .....	<b>ntPassFsLib</b>	198
close socket upload path	(Windview). .....	<b>sockUploadPathClose()</b>	1253
path to host using socket	(Windview). establish upload .....	<b>sockUploadPathCreate()</b>	1254
<b>wvSockUploadPathLib</b> library	(Windview). initialize.....	<b>sockUploadPathLibInit()</b>	1254
write to socket upload path	(Windview). .....	<b>sockUploadPathWrite()</b>	1255
close TSFS-socket upload path	(Windview). .....	<b>tsfsUploadPathClose()</b>	1402
path to host using TSFS socket	(Windview). open upload.....	<b>tsfsUploadPathCreate()</b>	1402
<b>wvTsfsUploadPathLib</b> library	(Windview). initialize.....	<b>tsfsUploadPathLibInit()</b>	1403
write to TSFS upload path	(Windview). .....	<b>tsfsUploadPathWrite()</b>	1403
log user-defined event	(WindView).....	<b>wvEvent()</b>	1469
instrument VxWorks Events	(WindView).....	<b>wvEventInst()</b>	1469
ID of WindView event buffer	(WindView). return.....	<b>wvEvtBufferGet()</b>	1470
events from those being logged	(WindView). clear class of.....	<b>wvEvtClassClear()</b>	1470
of events from those logged	(WindView). clear all classes .....	<b>wvEvtClassClearAll()</b>	1470
set of classes being logged	(WindView). get current.....	<b>wvEvtClassGet()</b>	1471
set class of events to log	(WindView).....	<b>wvEvtClassSet()</b>	1471
initialize event log	(WindView).....	<b>wvEvtLogInit()</b>	1472
start logging events to buffer	(WindView).....	<b>wvEvtLogStart()</b>	1472
stop logging events to buffer	(WindView).....	<b>wvEvtLogStop()</b>	1473
initialize <b>wvLib</b> - first step	(WindView).....	<b>wvLibInit()</b>	1473
initialize <b>wvLib</b> - final step	(WindView).....	<b>wvLibInit2()</b>	1473
create event-log header	(WindView).....	<b>wvLogHeaderCreate()</b>	1474
transfer log header to host	(WindView).....	<b>wvLogHeaderUpload()</b>	1474
reporting of network events to	WindView. end .....	<b>wvNetDisable()</b>	1476
reporting network events to	WindView. begin.....	<b>wvNetEnable()</b>	1476
instrument objects	(WindView).....	<b>wvObjInst()</b>	1481
object instrumentation on/off	(WindView). set.....	<b>wvObjInstModeSet()</b>	1482
of WindView rBuff manager	(WindView). set priority .....	<b>wvRBuffMgrPrioritySet()</b>	1483
instrument signals	(WindView).....	<b>wvSigInst()</b>	1483
extra copy of task name events	(WindView). preserve.....	<b>wvTaskNamesPreserve()</b>	1484
preserved task name events	(WindView). upload .....	<b>wvTaskNamesUpload()</b>	1485
register timestamp timer	(WindView).....	<b>wvTmrRegister()</b>	1485
start upload of events to host	(WindView).....	<b>wvUploadStart()</b>	1486
stop upload of events to host	(WindView).....	<b>wvUploadStop()</b>	1487

	Keyword	Name	Page
stack size of tWVUpload task	(WindView). set priority and .....	<b>wvUploadTaskConfig()</b>	1488
event logging control library	(WindView). .....	<b>wvLib</b>	351
timer library	(WindView). .....	<b>wvTmrLib</b>	357
set or display eventpoints	(WindView). .....	<b>e()</b>	567
close event-destination file	(WindView). .....	<b>fileUploadPathClose()</b>	592
file for depositing event data	(Windview). create .....	<b>fileUploadPathCreate()</b>	592
<b>wvFileUploadPathLib</b> library	(Windview). initialize .....	<b>fileUploadPathLibInit()</b>	593
to event-destination file	(WindView). write .....	<b>fileUploadPathWrite()</b>	593
(WindView). return ID of	WindView event buffer .....	<b>wvEvtBufferGet()</b>	1470
Interface Library.	WindView for Networking .....	<b>wvNetLib</b>	356
(WindView). set priority of	WindView rBuff manager .....	<b>wvRBuffMgrPrioritySet()</b>	1483
stream. read next	word (32-bit integer) from .....	<b>getw()</b>	640
stream. write	word (32-bit integer) to .....	<b>putw()</b>	1047
one buffer to another one long	word at a time. copy .....	<b>bcopyLongs()</b>	426
copy one buffer to another one	word at a time. ....	<b>bcopyWords()</b>	427
reset trigger	work queue task and queue .....	<b>trgWorkQReset()</b>	1400
create CBIO	wrapper atop BLK_DEV device. ....	<b>cbioWrapBlkDev()</b>	485
flush processor	write buffers to memory. ....	<b>cachePipeFlush()</b>	460
	write bytes to file. ....	<b>write()</b>	1468
output stream (ANSI).	write character to standard .....	<b>putchar()</b>	1046
(ANSI).	write character to stream .....	<b>putc()</b>	1045
(ANSI).	write character to stream .....	<b>fputc()</b>	610
register (MIPS).	write contents of cause .....	<b>intCRSet()</b>	687
device.	write data to SCSI tape .....	<b>scsiWrtTape()</b>	1168
sequential device.	write file marks to SCSI .....	<b>scsiWrtFileMarks()</b>	1167
do task-level	write for tty device. ....	<b>tyWrite()</b>	1412
buffer (ANSI).	write formatted string to .....	<b>sprintf()</b>	1256
	write formatted string to fd .....	<b>fdprintf()</b>	588
standard error stream.	write formatted string to .....	<b>printErr()</b>	996
standard output stream/	write formatted string to .....	<b>printf()</b>	997
stream (ANSI).	write formatted string to .....	<b>vfprintf()</b>	1431
stream (ANSI).	write formatted string to .....	<b>fprintf()</b>	606
(ANSI).	write from specified array .....	<b>fwrite()</b>	633
initiate asynchronous	write (POSIX). ....	<b>aio_write()</b>	410
block device.	write sector(s) to SCSI. ....	<b>scsiWrtSecs()</b>	1168
variable argument list to/	write string formatted with .....	<b>vsprintf()</b>	1446
variable argument list to fd.	write string formatted with .....	<b>vfdprintf()</b>	1430
variable argument list to/	write string formatted with .....	<b>vprintf()</b>	1446
output stream (ANSI).	write string to standard .....	<b>puts()</b>	1047
	write string to stream (ANSI). ....	<b>fputs()</b>	611
flash device.	write to boot-image region of .....	<b>tffsBootImagePut()</b>	1362
file (WindView).	write to event-destination .....	<b>fileUploadPathWrite()</b>	593
	write to non-volatile RAM. ....	<b>sysNvRamSet()</b>	1310
(Windview).	write to socket upload path .....	<b>sockUploadPathWrite()</b>	1255
(Windview).	write to TSFS upload path .....	<b>tsfsUploadPathWrite()</b>	1403
stream.	write word (32-bit integer) to .....	<b>putw()</b>	1047
table (68K, x86, ARM, /	write-protect exception vector .....	<b>intVecTableWriteProtect()</b>	704
(VxVMI).	write-protect text segment .....	<b>vmTextProtect()</b>	1445
from specified MTRR table with	WRMSR instruction. set MTRRs .....	<b>pentiumMtrrSet()</b>	957
set	WTX timeout. ....	<b>symSyncTimeoutSet()</b>	1295

	Keyword	Name	Page
(Windview). initialize	<b>wvFileUploadPathLib</b> library .....	<b>fileUploadPathLibInit()</b>	593
initialize	<b>wvLib</b> - final step (WindView). .....	<b>wvLibInit2()</b>	1473
initialize	<b>wvLib</b> - first step (WindView). .....	<b>wvLibInit()</b>	1473
(Windview). initialize	<b>wvSockUploadPathLib</b> library .....	<b>sockUploadPathLibInit()</b>	1254
(Windview). initialize	<b>wvTsfsUploadPathLib</b> library .....	<b>tsfsUploadPathLibInit()</b>	1403
status register (68K, MIPS,	x86). set task.....	<b>taskSRSet()</b>	1347
content of Control Register 2	(x86). get .....	<b>vxCr2Get()</b>	1447
value to Control Register 2	(x86). set.....	<b>vxCr2Set()</b>	1447
content of Control Register 3	(x86). get .....	<b>vxCr3Get()</b>	1448
value to Control Register 3	(x86). set.....	<b>vxCr3Set()</b>	1448
content of Control Register 4	(x86). get .....	<b>vxCr4Get()</b>	1448
content of Control Register 0	(x86). get .....	<b>vxCr0Get()</b>	1449
value to Control Register 4	(x86). set.....	<b>vxCr4Set()</b>	1449
value to Control Register 0	(x86). set.....	<b>vxCr0Set()</b>	1450
of Debug Register 0 to 7	(x86). get content.....	<b>vxDrGet()</b>	1450
value to Debug Register 0 to 7	(x86). set.....	<b>vxDrSet()</b>	1451
get content of EFLAGS register	(x86).....	<b>vxEflagsGet()</b>	1451
set value to EFLAGS register	(x86).....	<b>vxEflagsSet()</b>	1452
Descriptor Table Register	(x86). get content of Global.....	<b>vxGdtrGet()</b>	1452
Descriptor Table Register	(x86). /content of Interrupt .....	<b>vxIdtrGet()</b>	1453
Descriptor Table Register	(x86). get content of Local.....	<b>vxLdtrGet()</b>	1453
management mode (PowerPC, SH,	x86). get power.....	<b>vxPowerModeGet()</b>	1456
management mode (PowerPC, SH,	x86). set power.....	<b>vxPowerModeSet()</b>	1456
get content of TASK register	(x86).....	<b>vxTssGet()</b>	1460
set value to TASK register	(x86).....	<b>vxTssSet()</b>	1460
clear entry from cache (68K,	x86).....	<b>cacheArchClearEntry()</b>	446
handler for C routine	(x86). construct interrupt .....	<b>intHandlerCreateI86()</b>	689
disable interrupt stack usage	(x86). enable or .....	<b>intStackEnable()</b>	695
and gate selector	(x86). /gate type(int/trap),.....	<b>intVecGet2()</b>	699
type(int/trap), and selector	(x86). set CPU vector, gate .....	<b>intVecSet2()</b>	703
/interrupt lock-out level (68K,	x86, ARM, SH, SimSolaris,/ .....	<b>intLockLevelGet()</b>	693
/interrupt lock-out level (68K,	x86, ARM, SH, SimSolaris,/ .....	<b>intLockLevelSet()</b>	693
/exception vector table (68K,	x86, ARM, SimSolaris, SimNT). .....	<b>intVecTableWriteProtect()</b>	704
and/ set interrupt level (68K,	x86, ARM, SimSolaris, SimNT .....	<b>intLevelSet()</b>	690
/ (trap) base address (68K,	x86, MIPS, ARM, SimSolaris,/ .....	<b>intVecBaseGet()</b>	696
/ (trap) base address (68K,	x86, MIPS, ARM, SimSolaris,/ .....	<b>intVecBaseSet()</b>	697
get interrupt vector (68K,	x86, MIPS, SH, SimSolaris,/ .....	<b>intVecGet()</b>	698
set CPU vector (trap) (68K,	x86, MIPS, SH, SimSolaris,/ .....	<b>intVecSet()</b>	699
/handler for C routine (68K,	x86, MIPS, SimSolaris).....	<b>intHandlerCreate()</b>	688
register edi (also esi - eax)	(x86/SimNT). /contents of.....	<b>edi()</b>	568
contents of status register	(x86/SimNT). return.....	<b>eflags()</b>	568
compute remainder of	x/y (ANSI). .....	<b>fmod()</b>	598
compute remainder of	x/y (ANSI). .....	<b>fmodf()</b>	598
compute arc tangent of	y/x (ANSI). .....	<b>atan2()</b>	419
compute arc tangent of	y/x (ANSI). .....	<b>atan2f()</b>	420
create empty	zbuf. ....	<b>zbufCreate()</b>	1492
delete bytes from	zbuf. ....	<b>zbufCut()</b>	1492
delete	zbuf. ....	<b>zbufDelete()</b>	1493
duplicate	zbuf. ....	<b>zbufDup()</b>	1494
insert zbuf into another	zbuf. ....	<b>zbufInsert()</b>	1496

	Keyword	Name	Page
from buffer and insert into	zbuf. create zbuf segment .....	<b>zbufInsertBuf()</b>	1497
copy buffer data into	zbuf.....	<b>zbufInsertCopy()</b>	1498
determine length in bytes of	zbuf.....	<b>zbufLength()</b>	1499
get next segment in	zbuf.....	<b>zbufSegNext()</b>	1501
get previous segment in	zbuf.....	<b>zbufSegPrev()</b>	1502
send	zbuf data to TCP socket.....	<b>zbufSockSend()</b>	1507
receive data in	zbuf from TCP socket. ....	<b>zbufSockRecv()</b>	1505
receive message in	zbuf from UDP socket. ....	<b>zbufSockRecvfrom()</b>	1506
it to TCP socket. create	zbuf from user data and send.....	<b>zbufSockBufSend()</b>	1502
send it to UDP socket. create	zbuf from user message and.....	<b>zbufSockBufSendto()</b>	1503
	zbuf interface library. ....	<b>zbufLib</b>	359
insert	zbuf into another zbuf. ....	<b>zbufInsert()</b>	1496
split	zbuf into two separate zbufs. ....	<b>zbufSplit()</b>	1509
send	zbuf message to UDP socket.....	<b>zbufSockSendto()</b>	1508
determine location of data in	zbuf segment. ....	<b>zbufSegData()</b>	1499
determine length of	zbuf segment. ....	<b>zbufSegLength()</b>	1501
specified byte location. find	zbuf segment containing .....	<b>zbufSegFind()</b>	1500
insert into zbuf. create	zbuf segment from buffer and.....	<b>zbufInsertBuf()</b>	1497
initialize	zbuf socket interface library.....	<b>zbufSockLibInit()</b>	1505
	zbuf socket interface library.....	<b>zbufSockLib</b>	361
copy data from	zbuf to buffer. ....	<b>zbufExtractCopy()</b>	1495
split zbuf into two separate	zbufs. ....	<b>zbufSplit()</b>	1509
code using public domain	zlib functions. inflate .....	<b>inflateLib</b>	123