PERKIN-ELMER

OS/32
SYSTEM LEVEL
Programmer Reference Manual

48-040 R00

The Perkin-Elmer Corporation, Computer Systems Division
2 Crescent Place, Oceanport, New Jersey 07757

Printed in the United States of America

TABLE OF CONTENTS

FIGURES

TABLES

APPENDIXES

PREFACE

This manual describes privileged supervisor call (SVC) facilities and other operating system features intended for use by system programmers, system analysts, designers, engineers, and training instructors.

Chapter 1 presents an overview of the operating system and the software it supports. Chapter 2 is a guide to writing executive tasks (e-tasks). Chapter 3 explains how to use SVC 0, SVC 2, and SVC 7 facilities. Chapter 4 contains a functional description of the SVC interception feature recently developed for OS/32. The uniform vertical forms control (VFC) feature is described in Chapter 5 along with other device independent and dependent features.

The OS/32 System Level Programmer Reference Manual is one of three manuals replacing the OS/32 Programmer Reference Manual. New features described in this manual include VFC and SVC interception. Chapters 2 and 3 were formerly Chapters 7 and 8 of the OS/32 Programmer Reference Manual.

This manual applies to the OS/32 R06 software release and higher.

The following manuals can be used in conjunction with this manual:

MANUAL NAME	PUBLICATION NUMBER
OS/32 System Macro Library Reference Manual	48-006
32-Bit Systems Software User Documentation Summary	48-015
OS/32 Operator Reference Manual	48-030
OS/32 System Generation (SYSGEN) Reference Manual	48-037
OS/32 Supervisor Call (SVC) Reference Manual	48-038
OS/32 Application Level Programmer Reference Manual	48-039

For further information on the contents of all Perkin-Elmer 32-bit software manuals, refer to the 32-Bit Systems Software User Documentation Summary.

CHAPTER 1
OS/32 AND SUBSYSTEMS

1.1 INTRODUCTION

Perkin-Elmer OS/32 is a general purpose, event-driven operating system for Perkin-Elmer 32-bit computer systems. OS/32 is generated into a system through a system generator program that provides parameters for tailoring OS/32 to a specific installation. The combined hardware and software capabilities provide support for all phases of program and system development. OS/32 supports concurrent multiprogramming, with up to 252 user tasks (u-tasks) written in any of the supported languages. The program development features minimize the time and effort needed to test, debug, and integrate application programs and systems. The operating system can be tailored to support a wide variety of configurations, ranging from small dedicated systems to large, multiprocessor, shared memory systems. OS/32 also supports a command language that allows complex jobs to be performed with minimum operator intervention.

OS/32 incorporates a powerful interrupt handling capability at the task level. This capability permits a task to be interrupted during its normal execution sequence by a variety of hardware and software conditions.

The roll function, supported by OS/32, allows total memory requirements of (rollable) tasks within the system to exceed available task memory. Portions of a task that exceed available task memory are rolled out to disk until memory is available. In real time applications, rolling is commonly used to queue low priority tasks while tasks of higher priority are active. The roll eligibility of a task is established when linked; however, a task option is provided to prevent roll of a task when necessary; e.g., when the task must be able to respond to real time events.

A basic data communications facilities package is supplied with OS/32. This package also provides support for higher level Perkin-Elmer data communications products.

The scope and power of the operating system can be extended through these OS/32 companion products:

- o Multi-Terminal Monitor (MTM)
- o Reliance

MTM is a subsystem monitor that uses the subtasking capabilities of OS/32 to provide a time-sliced, interactive program development environment for up to 64 concurrent terminal users. MTM simultaneously supports both online terminal users and batch background tasks. MTM terminal users are also provided with an I/O spooler for use with slow speed devices.

Reliance is a transaction software system, consisting of the Integrated Transaction Controller (ITC), Data Management System (DMS/32), and industry standard COBOL. ITC allocates system resources, develops screen formats, and controls terminals. DMS/32 supervises disk allocation and data access.

1.2 SOFTWARE SUBSYSTEMS

OS/32 consists of the following subsystems:

- Task management
- Job accounting
- Memory management
- Timer management
- File management
- I/O management
- Error recording and reporting
- Memory diagnostics
- Loader and segmentation
- Basic communications
- Console monitor
- Command processor
- System initialization
- Internal interrupt
- Optional user supervisor calls (SVC)
- Floating point

Table 3-1 summarizes the software supported by OS/32.

TABLE 1-1 PERKIN-ELMER OS/32 SOFTWARE SUPPORT

TYPE	SOFTWARE PRODUCT	STD.	OPT.
Program Development	Task management	x	
	Job accounting	x	
	Memory management	x	
	Timer management	x	
	File management	x	
	I/O management	x	
	Error recording and reporting	x	
	Memory diagnostics	x	
	Loader and segmentation	x	
	Console monitor	x	
	Command processor	x	
	Floating point	x	
	Internal interrupt subsystem	x	
	*Integrated transaction controller (ITC)		x
	Writable control store (WCS)		x
	Multi-terminal monitor (MTM)		x
Data Base Management	*Data Management System (DMS/32)		x
Data Communications	Asynchronous data communications	x	
	Character synchronous communications	x	
	Bit synchronous communications		x
	2780/3780 RJE emulation		x
	3270 emulation		x
	HASP/32		x
Languages	Common microcode assembler (MICROCAL)		x
	Common assembly language (CAL)	x	
	CAL macro	x	
	FORTRAN VII Development (D) Language		x
	FORTRAN VII Optimizing (O) Language		x
	*COBOL		x
	Basic Level II		x
	Coral 66		x
	RPG II		x
	Pascal		x
Utilities	Link	x	
	Edit	x	
	Text		x
	Source Updater	x	
	AIDS (Automatic Interactive Debugging system)	x	
	Copy	x	
	Library loader	x	
	Macro library	x	
	Sort/Merge II		x
	Patch	x	

* ITC, COBOL, and DMS/32 comprise the Perkin-Elmer Reliance software system designed for transaction programming.

1.2.1 Task Management Subsystem

The task management subsystem provides all the functions required to schedule, coordinate, and supervise user applications. Task scheduling is provided on a user-defined priority basis that determines the order in which each task gains control of the processor. Tasks at the same priority level are serviced on a round-robin basis, which can optionally use time-slicing to limit the execution of any one task.

OS/32 recognizes 255 priority levels from a high of 1 to a low of 255. Of these levels, 10 through 249 are available to u-tasks, while 1 through 9 and 250 through 255 are reserved for system use. Each task can be given three priority levels:

- o Maximum
- o Task
- o Dispatch

Maximum priority, set by Link, is the highest priority level at which a task can be assigned. Task priority is the priority at which a task is currently assigned. Initially, task priority is set by Link but can be modified as a result of command execution. A dispatched task usually has a priority level equal to its task priority; however, if a higher priority task requires specific system resources being used by the current dispatched task, the dispatched task priority can be raised so that it can complete execution.

If a task in control of the CPU has the same priority as an incoming task, the CPU task remains active until it relinquishes control of the processor. Priorities should be assigned so that tasks that require a large amount of processor time do not lock out other tasks.

A task will relinquish control of the processor to another task when:

- o it is paused by the system operator;
- o it is cancelled by the system operator, user, or by another task;

- a higher priority task becomes ready due to an external event, such as the completion of an outstanding I/O request;
- it executes a macro call that places itself in a wait, paused, or dormant state; or
- it initiates I/O to a device.

Rather than being scheduled on a strict priority basis, tasks can be time-sliced. This option allows tasks of equal priority to receive equal shares of processing time. The operator command SET SLICE enables or disables the time-slicing option. When a task becomes ready, it is queued on a round-robin basis behind all ready tasks of equal priority.

The task manager maintains three chains to facilitate task scheduling:

- Rollin chain
- Ready chain
- Wait chain

The rollin chain is a queue containing a list of tasks to be rolled into the system and put on the ready chain for execution. The task manager removes tasks from the rollin chain and places them on the ready chain in priority order. Specific wait conditions must first be satisfied before a task is removed from the rollin chain and put on the ready chain.

The ready chain is a queue containing a linked list of task control blocks (TCB) that are ready to execute. A task must be removed from the rollin chain or wait chain and added to the ready chain before it can be dispatched for execution. Only tasks on the ready chain are eligible for execution. The ready chain is maintained according to task priority. The task with the highest priority is moved to the head of the ready chain and dispatched for execution. Tasks are scheduled for execution on a first-in/first-out (FIFO) basis within each priority level. If time-slicing is enabled, the task at the head of the ready chain relinquishes control when its time expires, or if an equal or higher priority task is ready. If no equal or higher priority task is ready, the task at the head of the ready chain continues to execute for another time-slice. If time-slicing is disabled, the task at the head of the ready chain continues executing until it voluntarily relinquishes control or until a higher priority task becomes ready.

The wait chain is a queue containing tasks that are in a wait state. These tasks require the completion of external events; e.g., I/O operations, before execution can be resumed. While the task is in the wait state, the task manager handles task queue entries using the task queue service method and also evaluates all pending and wait conditions. After all external events are satisfied, the task is in a ready state and is returned to the ready chain at the proper priority.

The task manager performs machine state changes through routines that switch or exit tasks from one machine state to another.

The task manager also performs context switching by saving the contents of registers into an alternate save area and exchanging the task status word (TSW).

1.2.2 Job Accounting Subsystem

The job accounting subsystem provides information such as CPU usage, elapsed time, utilized memory and disk space, and number and length of I/O transfers by device class. The job accounting subsystem contains the:

- o Data collection facility
- o Account reporting utility

The data collection facility collects accounting data on all user activities that occur in the system and stores information in the accounting transaction file (ATF) when the task terminates. Reports can be requested for individual user accounts, summaries of user accounts, and system usage. The account reporting utility is written in Perkin-Elmer FORTRAN VII and is designed to readily allow implementation of specific customer site requirements. The performance data collected by the data collection facility is prepared by the account reporting utility for use by system maintenance personnel.

Through the DISPLAY ACCOUNTING command, the system operator has access to accounting data for one or all tasks in the system.

1.2.3 Memory Management Subsystem

At load time the memory management subsystem dynamically allocates space for user and system applications. There are two types of memory:

- o Local
- o Global

Local memory is physically contiguous starting from location 0 and contains the operating system, task space, and system space.

Global memory is located above local memory and is not required to be contiguous. This area is shared by global task common segments. If global memory is located in multiport memory, it can be shared by more than one processor.

Memory is allocated on a first-fit basis when sufficient memory is available for a specific task. Free segments are allocated in ascending address order. When no space is available for a task, the memory manager determines which tasks are to be rolled out to ensure that high priority tasks take precedence. When memory becomes free, adjacent areas are merged together to minimize search time and to provide large free blocks of memory for bigger tasks. System task space is also maintained by the memory manager and is dynamically allocated when a task or device structure is built.

The memory manager maintains task space through free and allocated lists. Segments are allocated dynamically on the first-fit basis by searching the free lists. When free task space is allocated to a segment, it is removed from the free list and connected to the allocated list. Similarly, whenever a segment is released, its memory space is removed from the allocated list and connected to (or merged into) the free list.

1.2.4 Timer Management Subsystem

The timer management subsystem provides u-tasks with a set of timer management/maintenance services. These services control all time dependent functions; e.g., time-slicing, I/O, job accounting, file dating, through the universal clock.

The following timer queues are maintained by the timer management subsystem:

- o Time of day
- o Device timeout
- o Communications device timeout
- o Interval timer

There are several timer routines that service these queues. Entries are placed on the time of day queue and the interval timer queue as a result of supervisor call 2 (SVC 2) timer requests. The control blocks on the time of day queue are referred to as timer queue elements (TQEs). The interval timer queue has the same format as the time of day queue but is maintained as a separate queue.

The universal clock consists of a line frequency clock (LFC) and a precision interval clock (PIC). The LFC generates an interrupt every 8.3 milliseconds or 120 times per second. The PIC interrupts when a task's requested time interval has expired or at intervals of 4095 milliseconds, whichever is shorter. If the interval terminates or the time of day is reached, the TCE is removed from system space and an appropriate trap is generated.

1.2.5 File Management Subsystem

The file management subsystem coordinates the disk file requirements of all tasks. The file manager allows u-tasks to operate with disk files in a device independent fashion. In addition, the file manager provides disk dependent facilities.

Perkin-Elmer 32-bit computers support a wide variety of disk devices ranging from the 256kb floppy disk to the 256Mb (MSM300) disks. OS/32 distinguishes between permanent and temporary files, providing a temporary file facility for scratch work. Two file types are provided by the OS/32 file manager: contiguous and indexed.

To ensure the integrity of information stored on disk, the file manager interleaves disk access requests from different tasks by using file access protocols.

The file access protocol is a 5-stage process, providing operational flexibility at all phases of file usage. An abbreviated protocol is used for temporary files. I/O calls to these files are automatically intercepted by the file manager.

All I/O operations to files or devices are done to logical units (lu) rather than to a specific device or file. A file or device must be assigned by the user to an lu by a command, macro, or an SVC 7 prior to an I/O operation. SVC 7 file management handler provides the following interfaces to the user for file/device management:

- Allocation
- Assignment
- Change access privileges
- Renaming files
- Reprotecting files
- Closing files (deassignment)
- Deletion
- Checkpointing
- Fetch attributes

1.2.6 I/O Subsystem

The I/O subsystem provides a uniform programming interface between the task and external devices. I/O operations can occur in the following task modes:

- o Wait-only - execution is halted until data transfer is completed.
- o Proceed I/O - task continues execution during data transfer.
- o Halt I/O - allows the task to recall proceed I/O requests.
- o Unconditional proceed I/O - task connects immediately to an I/O device.
- o Queued I/O - task continues executing even though more than one proceed I/O calls are outstanding. A task trap mechanism reports each completed I/O operation individually. Wait-only and test I/O functions allow the task to synchronize its execution with the completed I/O operations.

1.2.7 Error Recording Subsystem

The error recording subsystem records on disk all internally logged errors, providing a data base for the Error Reporting Utility. This utility analyzes the recorded data and generates reports.

OS/32 memory error recording software supports the memory error log hardware of the Perkin-Elmer 3200 Series machines. Error log hardware keeps a history of the single bit corrected memory errors. The operating system reads the error log hardware and stores the error information into an internal error log buffer. When the error log buffer is full, its contents are stored on an error recording file with the date and time of the last error recorded. When the error recording file is almost full, a warning message is displayed on the system console indicating that a new error recording file should be allocated or that the Error Reporting Utility should be initiated. The Error Reporting Utility provides reports on the previously recorded error information in the error recording file.

The current error status can be displayed to the system console by using the DISPLAY ERRORS command. The internal error log readout period can be changed by the system operator.

1.2.8 Memory Diagnostics Subsystem

The memory diagnostics subsystem eliminates inoperable memory areas from the system configuration without affecting task execution. It enables the operating system to execute with part of real memory removed (holes) or powered down for maintenance purposes. Memory is tested, marked on, and marked off by using the operator MEMORY command and by the initialization of the operating system.

The marked off areas are noted in the memory map. Memory is marked on when previously marked off memory is to be used again. If an unrecoverable memory error on any Perkin-Elmer 3200 Series machine occurs during task execution, the operating system automatically marks off the area occupied by the task.

1.2.9 Loader and Segmentation Subsystem

The OS/32 resident loader is responsible for loading tasks, re-entrant libraries, task common (TCOM) segments, and shared segments. These tasks and segments must have been built by Link. Each load module generated by Link contains information related to the load module, e.g.; task options, size, libraries referenced. The OS/32 resident loader uses this information to generate data areas, set the task options, create segment tables for the u-tasks, and map all the necessary segments of the tasks. All u-tasks in OS/32 are built as though they were to be loaded at physical address 0 in memory. Through the use of the relocation/protection hardware, the task addresses are automatically relocated at run time by using the task segment table. This process is totally transparent to the user.

The loader is also responsible for creating the task environment, allocating roll files, creating, maintaining, and deleting segment tables, maintaining a segment control list, and mapping and unmapping shared segments.

The segmentation of tasks into pure and impure segments allows users to generate multiaccess or multithreading. Regardless of the number of times such a task is required for concurrent use, only one copy of the pure segment is loaded into memory by the operating system, whereas an impure segment is loaded for each invocation of the task. OS/32 assures the integrity of a pure segment by using the relocation/protection hardware to give read-only access to pure segments. Access to task common is achieved mnemonically in FORTRAN or assembly programs. The linkages are resolved by OS/32 Link, which also is used to request read, write, and execute privileges to task common.

1.2.10 Basic Data Communications Subsystem

The basic data communications subsystem provides a software interface between u-tasks and common carrier facilities. Basic data communications facilities enable the user to access remote terminals or computers as locally attached peripherals. For example, with OS/32 Communications there is no difference to the u-task between local teletype (TTY) and remote TTY access.

In addition to providing device independent (logical I/O) access to the u-task, the subsystem provides a set of more basic functions that allow systems to support special procedures, terminals, or facilities. Using the device dependent (physical I/O) capability, the system builder can tailor a communications package for a particular installation.

This package includes device dependent and device independent support of asynchronous line devices as well as device dependent support of binary synchronous lines. Terminal managers imbedded in the software provide high level control for TTY and video display units (VDU).

The OS/32 Basic Data Communications support package is required for all 32-bit communications products; e.g., 2780/3780 Remote Job Entry and the ZDLC Channel Terminal Manager, which supports SDLC, HDLC, and ADCCP protocols.

1.2.11 Console Monitor Subsystem

The console monitor subsystem processes all I/O requests directed to the system console device and the system log device from u-tasks and from the command processor task. The console driver is responsible for intercepting system console I/O requests and for directing them to the console monitor or to another monitor task such as MTM. All I/O from the system console directed to u-tasks running under MTM are routed through MTM and not through the console monitor.

When a command is issued from the system console to the command processor, the console monitor issues an SVC 6 to the command processor notifying it of a command to be processed. The command processor interprets the command and issues an SVC 6 to the console monitor indicating that it is ready to accept another command.

The console driver is a part of the OS/32 I/O subsystem and is the module that intercepts I/O requests to the system console, processes them, and gives them to MTM or to the console monitor to do the actual I/O.

The console monitor is the first task dispatched as a part of the OS/32 initialization phase. During this phase, the console monitor initializes both itself and the dummy device control block (DCB) used by the console driver to intercept requests from the system console. The monitor then issues an SVC 6 to start the command processor.

1.2.12 Command Processor Subsystem

The command processor subsystem accepts commands from the system console monitor task, decodes them, and calls the appropriate executor. The command processor contains logic to provide the console operator with informative messages in case of error.

An extension to the command processor, the command substitution system (CSS) allows commonly performed operations to be executed with one command. The CSS routines provide the ability to build, execute, and control files of operator and MTM commands. The user establishes command files that are called from the user console and executed in the user defined sequence. In this way, complex operations can be carried out by the user with few commands. These commands are analogous to macro instructions in assembly language. A set of logical CSS commands is provided to control the precise sequence of commands to be executed. Parameters can be passed to a CSS file so that general sequences can be written that take on specific meaning only when the parameters are substituted. Other calls to CSS can be imbedded within the CSS file (nested calls).

The command processor normally runs at the second highest priority level after the console monitor in OS/32. This task is strictly trap driven and responds to the SVC 6 task queue parameter calls from the console monitor to service a command request. When the command is processed, it signals the console monitor for a new command read via an SVC 6 queue parameter call and then suspends itself into a trap wait state. The command processor priority can be decreased by the operator ATTN command. This might be necessary in a real time application environment when some other task in the system has to run at a higher priority than the command processor.

1.2.13 System Initialization Subsystem

After the operating system is initialized, the system initialization subsystem initiates the memory diagnostics and error recording subsystems and establishes control blocks and tables in memory. It then dispatches the console monitor which readies the command processor to accept commands from the system console.

1.2.14 Internal Interrupt Subsystem

The internal interrupt sybsystem is responsible for:

- o handling illegal instruction faults,
- o handling arithmetic faults,
- o detecting memory faults,
- o handling system queue service interrupts,
- o handling protection/relocation hardware faults,
- o handling data format/alignment faults,
- o handling power fail and power restore conditions,

- restoring an interrupted task to its previous program state upon resumption of the task,
- handling parameter block errors,
- handling illegal SVCs and SVC interrupts,
- handling unrecoverable machine malfunctions, and
- taking memory image dumps.

Processor dependent interrupt handlers basically form the interrupt handler. The internal interrupt subsystem does not support external interrupts. External interrupts are handled directly by the appropriate device driver.

1.2.15 Optional User SVC Subsystem

SVC 14 is provided as an optional supervisory call that can be defined by the user. Upon execution, the task resumes a task trap for SVC 14.

1.2.16 Floating Point Subsystem

The u-task has optional access to single precision and/or double precision floating point instructions under OS/32. Floating point instructions can be executed through hardware or software. Those systems that do not support floating point options handle all floating point instructions as illegal instructions.

CHAPTER 2
EXECUTIVE TASKS (E-TASKS)

2.1 INTRODUCTION

Link classifies OS/32 tasks as user-tasks (u-tasks) or e-tasks. U-tasks run with the protection/relocation hardware enabled, while e-tasks run with the protection/relocation hardware disabled. Only e-tasks can add to or modify the system. For example, additions can be made by writing a driver to support a nonstandard peripheral device, and extensions can be made by modifying one of the system modules to include or enhance features.

Also described in this chapter are extended privileges specified through the Link OPTION command for u-tasks. These privileges allow direct task assignment to a disk device and file access by account number.

2.2 WRITING E-TASKS

An e-task, written as a u-task with some additional capabilities and restrictions, is executed in e-task state by specifying the Link OPTION E TASK command. In e-task state, no protection or memory relocation is provided; interrupts, except arithmetic fault, are enabled; and all machine instructions and addresses are allowed.

The system pointer table (SPT) provides access to system tables and control information. The address of the SPT is stored in the halfword location X'62' in memory.

2.3 RESTRICTIONS ON WRITING E-TASKS

Special care must be taken when writing an e-task. Since an e-task can be loaded into any area of memory and is run with the protection/relocation hardware disabled, it must be coded as positionally independent and must not contain relocatable code. Therefore, an e-task:

- o must not contain RX1 or RX3 instructions with relocatable addresses
- o must not assemble predefined address constants since addresses are relocated to absolute zero. For example:

```
SVC7BLK   DB    X'80',7
          DAC   ADDR
```

- must not be segmented. The e-task must dynamically set its parameter blocks; e.g., to reference address constants within a 16kb range, by using this technique:

```
          LA    UE,BUFSTART
          LA    UF,BUFEND
          LA    U3,SVC1BLK
          STM   UE,SVC1.SAD(U3)
          SVC   1,0(U3)
```

- must use the CAL NORX3 option. References exceeding a 16kb range can be made in this manner:

```
BASE          LA    U4,BASE
              LA    UE,BUFSTART-BASE(U4)
              LA    UF,BUFEND-BASE(U4)
              LA    U3,SVC1BLK-BASE(U4)
              STM   UE,SVC1.SAD(U3)
              SVC   1,0(U3)
```

2.4 DATA STRUCTURE MACRO LIBRARY

The data structure macro library is stored in file SYSSTRUC.MLB. These macros are used by e-tasks. Table 2-1 lists the data structure macro calls found in SYSSTRUC.MLB.

TABLE 2-1 DATA STRUCTURE MACRO CALLS USED BY EXECUTIVE TASKS (E-TASKS)

MACRO	DATA STRUCTURE
$ACB	Directory access control block
$CCB	Channel control block (CCB)
$DATB	Device attributes equates
$DCBS	$PDCB,$DDCB,DCB$EQUATE,$DFLAG,$DATB,$$DXFL
$DDCB	Device dependent device control block (DCB)
$DFLG	DCB flags
$DIES	Primary directory entry
$DXFL	Disk extended flags
$EREGS	16 executive registers (E1=register 1)
$EVN	Event node
$FCB	File control block (FCB)
$FCBS	FCB and FCB flags
$FDE	Free block descriptor entry

$FFLG$	FCB flags
$FD	File descriptor (fd)
$IOB	I/O block
IOB	I/O and I/O flags
$IOBF	I/O block flags
$IOH	I/O handler list
$IVT	Initial value table
$LIB	Loader information block (LIB)
LIB	LIB and loader options
$LOPT	Loader options
$LSG	Load segment
$LTCB	Loader task control block (TCB) redefinitions
$MAGDCB	Magnetic tape DCB
$MSMDCB	Mass storage module DCB
$PDCB	Primary (device independent) DCB
$PFCB	Private FCB
$PSW	Program status word (PSW)
$REGS$	$SOPT,$UREGS,$ERREGS,$RREGS,$PSW
$RLST	Roll selection list
$RREGS	16 general user registers (R1=register 1)
$RSARCPY	Reentrant system state alternate save area
$SDE	Segment descriptor element
$SOPT	System options
$SPT	System pointer table (SPT)
$SPTE	SPT extern definitions
$SPCL	Spooler message
$SVC1	Supervisor call 1 (SVC 1)
$SVC1$	SVC 1 and SVC 1 error codes
$SVC1ERR	SVC 1 error codes
$SVC4	SVC 4
$SVC5	SVC 5
$SVC6	SVC 6
$SVC7	SVC 7
$TCB	TCB
TCB	$TCB,$TOPT,$ISTT,$TWT,$SDE, and IOB
$TMQ	Timer queue entry (TQE)
$TOPT	Task options flags
$TSTT	Internal task status flags
$TSW	Task status word (TSW)
$TWT	Task wait status flags
$UDL	User dedicated locations (UDL)
UDL	UDL and TSW
$UREGS	16 general user registers (U1=register 1)
$VD	Volume descriptor

2.5 SYSTEM EXTENSIONS

The operating system can be extended or modified by incorporating changes into the source of one or more Perkin-Elmer supplied system modules or by adding a user-written system module to the system. All data structures should be referenced by calling the data structure (macro) from the data structure macro library at assembly time and by using field definitions in all instructions referencing the structure.

2.6 TASK ENVIRONMENTS

OS/32 provides three task environments:

- o Universal
- o Foreground
- o Background

Link assigns a task to a universal environment. Universal tasks loaded from the system console can communicate with tasks in the foreground environment as well as other universal tasks. A universal task cannot be loaded as a background task or from a multi-terminal monitor (MTM) terminal that has the intertask communication option disabled.

Foreground tasks have the full range of OS/32 services available to them. Background tasks are identified when loaded with the taskid .BG. Foreground and background tasks cannot communicate with one another.

To prevent background tasks from interfering with foreground tasks, a background task is restricted as follows:

- o The maximum priority level and system space of a background task are set at system generation (sysgen) time.
- o A background task cannot communicate with tasks in other environments and vice versa. Intertask communication and control functions are treated as no-ops or illegal calls according to the task options chosen.
- o A background task cannot access task common segments.

CHAPTER 3
SYSTEM LEVEL SUPERVISOR CALLS (SVC)

3.1 INTRODUCTION

This chapter describes the functions and capabilities of SVCs used by executive tasks (e-tasks) and privileged user tasks (u-tasks).

3.2 SVC 0

SVC 0 is reserved for user-written SVC calls. Modification of the operating system is necessary in order to write this SVC call.

3.3 SVC 2 FACILITIES

Command processors are a part of utility programs and of many applications programs. Every time a program is written to accept commands, either by conversation with an operator or by command substitution system (CSS) commands, a command processor must interpret those commands. Most command processors are so similar that one could be put together from a canned routine package.

This chapter guides the user when using SVC 2 calls provided in OS/32 for command processing functions.

3.3.1 SVC 2 Code 0: Make Journal Entries

SVC 2 code 0 makes an entry into the system journal from an e-task. The system journal provides a method to trace back important events (SVC calls, I/O operation, task switching) that occurred during system operation. The journal is useful for tracing the cause of a system failure. This is accomplished through the SVC 2 code 0 parameter block shown in Figure 3-1.

0(0) Code	2(2) Journal code
4(4) Value 1	

```
|-------------------------------------------------------------|
|8(8)                                                          |
|                          Value 2                             |
|                                                              |
|-------------------------------------------------------------|
|12(C)                                                         |
|                          Value 3                             |
|                                                              |
|-------------------------------------------------------------|
|16(1C)                                                        |
|                          Value 4                             |
|                                                              |
 -------------------------------------------------------------
```

```
                SVC     2,parblk
                 .
                 .
                 .
        parblk  DC      H'0'
                DC      H'journal code'
                DC      F'value 1'
                DC      F'value 2'
                DC      F'value 3'
                DC      F'value 4'
```

Figure 3-1 SVC 2 Code 0 Parameter Block Format and Coding

During execution, a logical OR operation is performed on the mask and the journal code to ensure that the entry originates from an SVC 2 code 0, and not from within the system. The values 1, 2, 3, and 4 fields of the parameter block are stored following the journal code and calling task name in the journal. These values can contain any desired information to be preserved for system debugging.

3.3.2 SVC 2 Code 16 (Format 2)

SVC 2 code 16 (format 2) changes a user-specified file descriptor (fd) in an e-task or privileged u-task from an unpacked format to a packed format. This operation is accomplished through the SVC 2 code 16 parameter block. If format 2 is executed in a u-task, the pack fd operation occurs, but the result cannot be used in a subsequent SVC 7 operation.

Format 2 of SVC 2 code 16 can be used only by a privileged u-task. The u-task becomes privileged if the account privilege option was specified in the OPTION command at link time. The account privilege option gives u-tasks the privilege to access files by specifying an account number instead of a file class. However, if the task loader has the e-task load option prevented, the privileged u-task is loaded into memory with the account privilege option changed to the default no account privilege. Therefore, access by account number cannot be performed.

The pack fd operation in format 2 is the same as in format 1 except for the following:

- o The fd in format 2 differs from that of format 1 in the file class portion. The fd for format 2 is:

```
voln:filename.ext/actno
dev
```

  The actno is a decimal number from 0 through 255 representing the user account number.

- o The SVC 2 code 16 parameter block option field is the same as that for format 1. The hexadecimal values for the option field differ from those in format 1, but their meanings are the same. Since the meanings for these options are the same as those in format 1, refer to the more detailed description under the format 1 options. The options for format 2 are:

OPTION	MEANING
X'08'	Default volume is system volume
X'48'	Default volume is system volume; skip leading blanks
X'28'	Default volume is spool volume
X'68'	Default volume is spool volume; skip leading blanks
X'88'	No default volume
X'C8'	No default volume; skip leading blanks

- o The area receiving the packed fd is the same as format 1, with the exception of the file class field. The format of the packed fd area used in format 2 is shown in Figure 3-2.

```
 -----------------------------------------------------------
|0(0)                                                       |
|               Volume name or device name                  |
|                                                           |
|-----------------------------------------------------------|
|4(4)                                                       |
|                                                           |
|                                                           |
|---------------          Filename          ----------------|
|8(8)                                                       |
|                                                           |
|                                                           |
```

12(C)	15(F)
Extension	Account number

Figure 3-2 Packed File Descriptor (fd) Area for Format 2

The 1-byte account number field receives the packed format of the user-specified account number. The account number in the unpacked fd is specified as a decimal number ranging from 0 through 255.

After executing the pack fd operation, the user-specified account number is represented as a hexadecimal value ranging from X'00' through X'FF', and the condition code is set to 2 (G bit set). If no account number is specified in the unpacked fd, an S is returned in the account number field of the packed fd when running under the operating system; a P is returned in the account number field of the packed fd when running under multi-terminal monitor (MTM). If one of the three file classes in format 1 is specified as the account number in the unpacked fd, that file class is returned in the account number field of the packed fd, and the G bit is not set.

The condition code settings for format 2 are the same as for format 1, with the addition of the G bit setting. The condition codes are:

C	V	G	L	
0	0	0	0	Normal execution
0	0	0	1	No volume name present in unpacked fd.
0	0	1	0	A numeric account number present in account number field.
0	1	0	0	Syntax error present in unpacked fd.
1	0	0	0	No extension present in unpacked fd.

3.3.3 SVC 2 Code 26: Fetch Device Name

SVC 2 code 26 searches for a user-supplied volume name in the volume mnemonic table and returns the device name on which that volume is mounted. The operation is accomplished through the SVC 2 code 26 parameter block shown in Figure 3-3.

```
------------------------------------------------------------------
|0(1)           |1(1)             |2(2)   User     |3(3)   User    |
|   Reserved    |      Code       | register 1     | register 2    |
------------------------------------------------------------------
```

```
          SVC    2,parblk
          .
          .
          .
parblk    ALIGN  4
          DB     0,26
          DB     user register number
          DB     user register number
```

Figure 3-3 SVC 2 Code 26 Parameter Block Format and Coding

This parameter block is four bytes long, fullword boundary aligned, and does not have to be in a writable segment of the task. The fields are described as follows:

Fields:

Reserved — is a 1-byte field that must contain a value of 0 to indicate no options for the call.

Code — is a 1-byte field that must contain the decimal value of 26 to indicate code 26 of an SVC 2.

User register 1 — is a 1-byte field that must contain a user-specified register number. The specified register contains a pointer to a fullword containing a 4-character volume name.

User register 2 — is a 1-byte field that must contain a user-specified register number. The specified register contains the address of the area receiving the device name. This area is 4 bytes long, fullword boundary aligned, and must be located in a task writable segment.

Possible condition codes occurring after SVC 2 code 26 execution are:

C	V	G	L	
0	0	0	0	Normal termination
0	1	0	0	Specified volume offline; no fetch occurred.

Example:

```
                .
                .
                LA     R1,MIMVCIN
                LA     R2,MIMDEVN
                SVC    2,FTCHDEVN
                .
                .
                .
                ALIGN  4
       FTCHDEVN DB     0,26
                DB     R1
                DB     R2
```

3.4 SVC 7

All SVC 7 functions used by u-tasks are available to e-tasks and privileged u-tasks. SVC 7 provides the following additional capabilities to e-tasks and privileged u-tasks.

3.4.1 Assign, Reprotect, and Rename Functions

An e-task and a privileged u-task can bypass the file manager and directly assign I/O requests to a disk device. When an e-task or privileged u-task issues an SVC 1 I/O request directly to a disk device, the operation is referred to as bare disk I/O and should always be random access. The supported I/O functions are read, write, test, and set.

Direct assignment to a disk device can be performed only by a privileged u-task. A u-task becomes privileged if the disk privilege option was specified in the OPTION command at link time. The disk privilege option gives u-tasks the privilege to bypass the file manager and directly assign to a disk device. However, if the task loader has the e-task option prevented, the privileged u-task is loaded into memory with the disk privilege option changed to the default no disk privilege. Therefore, bare disk I/O cannot be performed by the task.

There are two types of direct disk assignments: online and offline. If the disk is marked online, only assignments for shared read only (SRO) are allowed. Any other access privilege specified at assignment time is rejected, and this message is displayed:

```
PRIV-ERR
```

If the disk is marked offline, all access privileges are allowed.

An e-task can reprotect both devices and files. The protection of a device or file can be changed to or from X'FF' (unconditional protection). An e-task does not have to specify keys when assigning a protected device or file. When an e-task issues an SVC 7 rename function, the e-task can change the name of a device.

3.4.2 Extended Close Function

The extended close function X'FF80' closes the specified logical unit (lu) and replaces the date and time of allocation and the last write (or write filemark) operation in the disk directory with the information from the parameter block. The format is shown in Figure 3-4.

0(0) Function code	2(2) Error status	3(3) lu
4(4) Allocation date/time (moved into DIR.DATE)		
8(8) Last write operation date/time (moved into DIR.LUSE)		

Figure 3-4 SVC 7 Parameter Block for Extended Close Functions

If a u-task executes an extended close function, the error code X'01' is returned to the parameter block error status field. The format of the date/time field in the parameter block must be the same format generated by the routine DATE.DIR. This subroutine is located in the file manager utility (FMUT) module. These functions can be executed through an SVC 7. No corresponding operator command exists for this call.

3.4.3 Extended Fetch Attributes Function

The extended fetch attributes function fetches the attributes of a bare disk device through its assigned lu. The write attribute is reset in the attributes halfword field (SVC7.KEYS) of the parameter block if the disk is marked on protected. The following device dependent information is returned to the parameter block:

- o The physical address of the controller is returned to the index block size (SVC7.ISZ) field.
- o The physical address of the channel is returned to the data block size (SVC7.DSZ) field.

3.5 SVC 6 RELEASE

For system tasks (.CMDP, .CSL, .MTM, and .SPL) the SVC 6 release function allows the user to specify the address of the instruction the task should execute after it is released. This address is stored in the SVC6.SAD field. If this field contains 0, the task continues executing with the instruction following the instruction executed before the task was suspended.

CHAPTER 4
SUPERVISOR CALL (SVC) INTERCEPTION

4.1 INTRODUCTION

SVC interception software is used to write programs that can emulate the SVC processing ability of OS/32. This software consists of macros that allow a task (intercepting task) to intercept the SVC of another task before it goes to the operating system for processing. Once interrupted, the SVC can be monitored by the intercepting task and sent to the operating system for processing, or it can be placed under the control of the intercepting task for processing. Table 4-1 lists the system macros used for SVC interception.

TABLE 4-1 SYSTEM MACROS FOR SUPERVISOR CALL (SVC) INTERCEPTION

MACRO	FUNCTION
ICREATE	Creates an SVC intercept path
IREMOVE	Removes a previously created path
IGET	Gets data from a data area of the application task that issued an intercepted SVC
IPUT	Puts data into a data area of the application task that issued an intercepted SVC
ICONT	Continues standard execution of an intercepted SVC by passing control to an OS/32 SVC executor
IPROCEED	Allows the application task that issued the intercepted SVC to proceed with its execution
IROLL	Makes an intercepted task rollable
ITERM	Terminates an intercepted SVC after processing
ITRAP	Sends a task queue item to a task
IERRTST	Evaluates errors returned by any of the above macros and branches execution to specific error routines within the intercepting task

The intercepting task tells the OS/32 SVC executor which SVC it will control or monitor. When the intercepting task is sent an SVC from the executor, the intercepting task executes the SVC while the application task that issued the SVC is placed in a wait state. The intercepting task can read from or write to the address space of the application task while executing the intercepted SVC.

The application task is not aware that its SVC has been intercepted unless it is informed by the intercepting task.

A task can intercept SVC calls only after it is linked with the intercept option. Once linked to the SVC interception software, the task can be programmed to intercept any of the following SVCs issued by any application task in the system:

- o SVC 1
- o SVC 2 code 7
- o SVC 3
- o SVC 6
- o SVC 7

4.2 HOW SVC INTERCEPTION WORKS

In general, SVC interception software functions as follows:

1. A task with SVC interception enabled by Link is built. This intercepting task must:

    - reserve memory for a set of request descriptor block (RDB) buffers for each SVC to be intercepted,
    - build a circular list for storing addresses of RDB buffers containing information of intercepted SVCs,
    - create intercept paths that designate which SVCs are to be intercepted (via ICREATE macro), and
    - define what control the intercepting task has over the SVCs it intercepts (via ICREATE macro).

2. An application task issues an SVC.
3. If no intercept path was created for that particular SVC, one of the standard OS/32 executors services the SVC.
4. If an intercept path has been created:

- the SVC is intercepted,

- information identifying the SVC is obtained and stored in a buffer whose address is removed from the circular list containing the address of free RDB buffers, and

- the intercepting task is notified that an SVC is ready for processing by a task event trap. When the task event trap service routine starts executing, register 1 contains the address of the RDB associated with the intercepted SVC, and register 0 contains the intercept path identifier that was supplied by the system when the intercept path was created. (To exit from the task event trap service state, the intercepting task issues a TEXIT macro.)

5. If the intercepting task intercept path had been built to monitor this SVC, the SVC is returned to the operating system for normal execution.

6. If the intercepting task has chosen to service the SVC, it is now ready to do so by issuing the intercept macros IGET, IPUT, IROLL, and ITRAP. Also, the intercepting task can issue the IPROCEED macro to allow the application task to continue executing during SVC processing.

7. After the intercepting task processes the SVC, the intercepting task issues an ITERM macro that transfers control back to the application task.

4.3 PREPARING TO CREATE AN INTERCEPT PATH

Before creating an intercept path, an intercepting task must:

o have a set of RDB buffers for each type of SVC to be intercepted, and

o build a circular list to store the addresses of the RDB buffers.

The size of each RDB buffer depends on the size of the parameter block for the particular SVC type. For example, a set of buffers allocated for SVC 6 interception will be larger than a set of buffers for SVC 1 interception. When an intercepting task uses one set of buffers for intercepting two or more SVC types, buffer size must equal the size of the RDB needed to hold the largest parameter block of the particular SVC types. Figure 4-1 shows the RDB fields. To define a structure containing these fields, use the $RDB macro.

```
 ---------------------------------------------------------------
|0(00)                          |2(02)   Interception           |
|           Reserved            |         identifier            |
|          (RDB.RID)            |         (RDB.PID)             |
|---------------------------------------------------------------|
|4(04) Parameter block offset   |6(06) SVC    |7(07)  Task      |
|          (RDB.OFF)            |    type     |   priority      |
|                               |  (RDB.SVC)  |  (RDB.TPRI)     |
|---------------------------------------------------------------|
|8(08)                                                          |
|             Operating system task identifier                  |
|                        (RDB.PAD)                              |
|---------------------------------------------------------------|
|12(0C)                                                         |
|               SVC parameter block address                     |
|                        (RDB.TID)                              |
|---------------------------------------------------------------|
|16(1C)          Instruction address following                  |
|                 intercepted SVC instruction                   |
|                        (RDB.SVAD)                             |
|---------------------------------------------------------------|
|20(14)                                                         |
|                     SVC parameter block 1                     |
|                        (RDB.PB)                               |
~                           .                                   ~
                            .
~                           .                                   ~
|              nn (nn)  SVC parameter block n                   |
|                                                               |
|                                                               |
 ---------------------------------------------------------------
```

Figure 4-1 Request Descriptor Block (RDB)

The fields contained within the RDB are described as follows:

Fields:

Reserved	is a halfword field reserved for future use.
Interception identifier (RDB.PID)	is a halfword field containing an SVC interception path identifier exclusively reserved for one particular SVC interception.
Parameter block offset (RDB.OFF)	is a halfword field containing the hexadecimal hexadecimal offset value for the parameter block within the RDB.

SVC type (RDE.SVC)	is a 1-byte field containing a decimal number specifying the type of SVC that is to be intercepted. - 01 indicates SVC 1 - 02 indicates SVC 2 (parameter block contains a decimal value of 7, indicating an SVC 2 code 7) - 03 indicates SVC 3 - 06 indicates SVC 6 - 07 indicates SVC 7
Task priority (RDE.TPRI)	is a 1-byte field containing a decimal number specifying the priority of the application task that issued the intercepted SVC.
Operating system task identifier (RDE.TID)	is a 4-byte field containing the operating system task identifier for the application task that issued the intercepted SVC.
SVC parameter block address (RDE.PAD)	is a 1-byte field containing a hexadecimal number specifying the address of the SVC parameter being intercepted. For SVC 3 interceptions, this field contains the end of task code.
Instruction address following intercepted SVC instruction (RDE.SVAD)	is a 1-byte field containing a hexadecimal number specifying the address of the instruction following the intercepted SVC instruction in the application task. This field is set to 0 for SVC 3 interceptions.
SVC parameter block 1 (RDE.PB)	is a variable length field containing the parameter block of the intercepted SVC.

The intercepting task must have a standard Perkin-Elmer circular list to hold the address of each RDB buffer. Figure 4-2 shows the fields of the standard circular list. When an SVC is sent to the intercepting task for processing, one RDB buffer address is automatically removed from the circular list, and the RDB is filled with information identifying the intercepted SVC. Refer to Perkin-Elmer Model 3220 Processor User's Manual, Publication Number C29-693 and Perkin-Elmer Model 3240 Processor User's Manual, Publication Number C29-685, for a more detailed explanation of the standard circular list.

```
 -------------------------------------------------------------
|0(00)                        |2(02)                          |
|          Number             |        Current number         |
|        of buffers           |          of buffers           |
|-------------------------------------------------------------|
|4(04)                        |6(06)                          |
|        Current top          |          Next bottom          |
|                             |                               |
|-------------------------------------------------------------|
|8(08)                                                        |
|                        A (buffer 1)                         |
|                                                             |
|-------------------------------------------------------------|
|12(0C)                                                       |
|                        A (buffer 2)                         |
|                                                             |
|-------------------------------------------------------------|
|                                                             |
~                             .                               ~
                              .
~                             .                               ~
|                                                             |
|-------------------------------------------------------------|
|                                                             |
|                        A (buffer n)                         |
|                                                             |
 -------------------------------------------------------------
```

Figure 4-2 System Task Buffer List (Standard Circular List)

Fields:

Number of buffers	is a halfword field indicating the number of fullwords in the entire list.
Current number of buffers	is a halfword field indicating the number of fullwords currently in use. When this field equals 0, the list is empty. When this field equals the number of fullwords in the list, the list is full.
Current top	is a halfword field indicating the address of the RDB buffer that is currently at the top of the list.
Next bottom	is a halfword field indicating the address of the next RDB buffer that is at the bottom of the list.
A (buffer n)	indicates the address of the RDB buffer.

4.4 CREATING INTERCEPT PATHS (ICREATE)

Before an intercepting task can execute an SVC, it must create a path to the application task that is to have an SVC intercepted. It does this through the ICREATE macro that informs the SVC executor which SVC is to be intercepted by this path. The intercepting task also accesses the application task address space through the intercept path.

An intercept path remains in effect until it is removed by the intercepting task creating it or until the application task terminates. Although only one type of SVC can be intercepted by each path, there is no limit to the number of paths that can be created by one intercepting task.

The mode parameter of the ICREATE macro specifies when an SVC is to be intercepted. Under caller mode, the specified SVC is intercepted every time it is issued from an application task. When the recipient existent mode is specified, the SVC is intercepted only when it is directed towards a specified task, device, pseudo task, or pseudo device that already exists in the system. Under the recipient nonexistent mode, the SVC is intercepted only when it is directed toward a specified pseudo task or pseudo device created by the ICREATE macro.

4.5 HOW TO CREATE A PSEUDO DEVICE OR TASK WITH ICREATE

A pseudo device consists of a name and the SVC 1 or SVC 7 intercept paths attached to it. The pseudo device name, which is known to the system but does not actually refer to any system device or file, consists of a device name, filename, and extension. Use a device name that does not already exist for a real device or disk volume. Pseudo devices ignore the account number field.

When the operating system searches for a device or filename and cannot find it in the system, it will search the list of pseudo devices. If a match occurs, the system will continue processing the SVC using the pseudo device.

To create a pseudo device using SVC interception software, set the ICREATE macro for either an SVC 1 or SVC 7 and specify the recipient nonexistent mode. An SVC 1 intercept path must be in effect if an I/O operation is attempted to a pseudo device; otherwise, an invalid function (X'C0') error status is returned.

A pseudo task consists of a name attached to one or more SVC 6 intercept paths. A pseudo task name is known to the system but does not refer to an actual task already existing in the system.

To create a pseudo task, issue the ICREATE macro specifying SVC 6 and the recipient nonexistent mode. Because a pseudo task does not refer to a real task, the pseudo task cannot be cancelled. Both pseudo tasks and pseudo devices are deleted by removing all intercept paths attached to them.

4.6 USE OF GENERIC NAMING FOR PSEUDO DEVICES AND TASKS

A pseudo device or task can be generically named. The following characters can be used for generic naming:

- o An asterisk (*) represents any character or blank.
- o A backward slash (\) represents any character.

If a pseudo device or task name specifies the filename and extensions fields as blanks, the system substitutes filename and extension fields filled with asterisks. This has the effect of generically naming the filename and extension fields so that they will always match the input filename and extension.

Generic naming cannot be used to name pseudo devices or tasks using the ICREATE macro. If recipient existent mode is specified with a generic pseudo device or task name, a pseudo device or task must exist with its name exactly matching the one specified by ICREATE, or an error will result. For example, a system is asked to create the following pseudo devices:

- o FAKE:FILE1
- o FAKE:FILE*
- o FAKE:
- o FAKE:FILE*.EXT

Normally, the following input will match the above pseudo devices:

INPUT NAME	SELECTED PSEUDO DEVICE
FAKE:	FAKE:
FAKE:FILE3	FAKE:FILE*
FAKE:FILE1	FAKE:FILE1
FAKE:FILE11	FAKE:
FAKE:FILEX.EXT	FAKE:FILE*.EXT
FAKE:FILEX.EX	FAKE:

When the ICREATE macro is issued specifying recipient nonexistent mode and the pseudo device FAKE:, ICREATE will not be executed because the pseudo device already exists. Consequently, when an ICREATE macro is issued specifying recipient existent mode along with the pseudo device FAKE:FILE*, ICREATE will be executed because the pseudo device FAKE:FILE* already exists.

4.7 FUNCTIONAL SUMMARY OF SVC INTERCEPTION

The following describes how interception works for each SVC and mode:

- o SVC 1 caller - Any SVC 1 issued by the specified task is intercepted.
- o SVC 1 recipient existent - Any SVC 1 directed to an lu assigned to the specified device or pseudo device is intercepted. (Note that disk volume interception is not supported for SVC 1.)
- o SVC 1 recipient nonexistent - The pseudo device is created, and any SVC 1 calls specifying an lu assigned to this pseudo device are intercepted.
- o SVC 2 code 7 caller - Any SVC 2 code 7 issued by the specified task is intercepted.
- o SVC 2 code 7 recipient existent and nonexistent - These calls are invalid.
- o SVC 3 caller - If the specified task goes to end of task for any reason, an SVC 3 intercept will occur.
- o SVC 3 recipient existent and recipient nonexistent - These calls are invalid.
- o SVC 6 caller - Any SVC 6 issued by the specified task is intercepted.
- o SVC 6 recipient existent - Any SVC 6 directed to the specified task or pseudo task is intercepted.
- o SVC 6 recipient nonexistent - The pseudo task is created, and any SVC 6 calls directed to this pseudo task are intercepted.
- o SVC 7 caller - Any SVC 7 issued by the specified task is intercepted.
- o SVC 7 recipient existent - Any SVC 1 directed to the specified device, disk volume, or pseudo device is intercepted.
- o SVC 7 recipient nonexistent - The pseudo device is created, and any SVC 7 calls specifying this pseudo device are intercepted.

4.8 FULL AND MONITOR CONTROL INTERCEPT PATHS

The ICREATE macro specifies the level of control that the intercept path allows an intercepting task to have over an application task.

A full control intercept path allows the intercepting task to exert full control over an application task whose SVC has been intercepted. Specifically, the intercepting task can:

- make the application task rollable via the IROLL macro. When an SVC is intercepted, the application task issuing the SVC is placed in a wait state and made unrollable. At the discretion of the intercepting task, the application task can be made rollable (assuming that the application task is able to be rolled).

- allow the application task to execute while it processes a proceed SVC via the IPROCEED macro. When an SVC is intercepted, the application task is placed in a wait state and made unrollable. At the discretion of the intercepting task, the application task can proceed with its execution while the intercepting task processes the SVC.

- obtain data from the application task memory space via the IGET macro.

- write data into the writable memory space of the application task via the IPUT macro.

- send a task queue item to the application task via the ITRAP macro. While processing the SVC, the intercepting task may find it necessary to send a task queue item to the application task. The task queue item sent must have a valid OS/32 reason code in the high order byte. In addition, the application task TSW must not have the task queue entry bit associated with the reason code disabled.

A monitor control intercept path intercepts an SVC to inform the intercepting task that the application task has issued that call. Once the intercepting task is aware that the SVC is ready for execution, the SVC is sent to the operating system for normal processing.

The following guidelines should be followed when assigning a level of control to the intercept path:

- Only monitor control can be specified for SVC 3 intercept paths. Either full or monitor control can be specified for all other SVC type intercept paths.

- Only one full control intercept path can be attached to a device or task (or pseudo device or task) for each type of SVC to be intercepted.

- A task or device (or pseudo task or device) can be attached to any number of monitor control intercept paths.

Example:

```
ICREATE   NAME=MAG:, MODE=RX,CONTROL=FC,
           SVC=(7)
ICREATE   NAME=MAG:, MODE=RX,CONTROL=FC,
           SVC=(1)
ICREATE   NAME=MAG:, MODE=RX,CONTROL=MC,
           SVC=(7)
ICREATE   NAME=MAG:, MODE=RX,CONTROL=MC,
           SVC=(1)
```

In this example, a full control SVC 7 intercept path is attached to device MAG:. A full control SVC 1 intercept path is also attached to MAG:. No other SVC 1 or SVC 7 full control intercept paths can be attached. Of course, any number of SVC 1 and SVC 7 monitor control intercept paths can be attached to MAG:; here one SVC 7 and one SVC 1 monitor control paths are attached.

4.9 HOW INTERCEPT PATHS HANDLE SVCS OCCURRING AT END OF TASK

SVC 1 and SVC 7 can be intercepted during end of task processing (including end of task processing after cancel), if intercept paths exist from these SVCs to devices assigned to the task's logical units. The intercepting task must be careful when writing into the operating system address space when executing these SVCs so as not to destroy the system's integrity.

If the application task is cancelled while the intercepting task is processing the SVC, SVC processing is aborted and the application task proceeds to the end of task.

4.10 TERMINATING THE INTERCEPTED SVCS

When the intercepting task receives an SVC from a full control intercept path, the intercepting task has the option of returning the SVC to the operating system for processing. To do this, the intercepting task issues an ICONT macro that allows the operating system to resume processing the intercepted SVC as if the intercept had never occurred. The ICONT macro cannot be used if an IPROCEED or IPOLL macro has been issued to the application task.

If the intercepting task chooses to process the SVC, the intercepting task issues an ITERM macro after the SVC is processed. ITERM terminates the interception and, if no IPROCEED has been issued, allows the application task to resume execution with the instruction immediately following the intercepted SVC instruction.

Either ICONT or ITERM can be used to terminate interception along a monitor control intercept path. The system does not differentiate between the two calls in this case. Here the ICONT or ITERM macro replaces the BDB buffer address back on to the circular list. It is very important that the ICONT or ITERM macro be used to replace the BDB.

Cancelling an application task under monitor or full control aborts the processing of the intercepted SVC in progress. The intercepting task must still issue an ICONT or ITERM to terminate the SVC interception.

4.11 HOW TO REMOVE INTERCEPT PATHS

An intercepting task can remove an intercept path by issuing an IREMOVE macro specifying the path to be removed. IREMOVE can be used for both immediate and delayed termination depending on whether the controlled shutdown or abort option is chosen.

The controlled shutdown option refuses all incoming requests and completes the servicing of all existing queued and executing SVCs. When processing of the last existing SVC intercepted by the path is completed, the path is removed from the system.

The abort option terminates all existing queued and executing SVCs before removing the intercept path from the system.

4.12 ERROR HANDLING

Run time errors that result from executing intercept macros are handled by user-written error routines within the intercepting task. When an error occurs, execution branches to the routine specified by either the IERRIST macro or the error parameter associated with each macro.

The IERRIST macro is written immediately after a macro for which the error parameter has been omitted. If an error occurs, execution of the intercepting task will branch to a user-written error routine to handle the error. Error codes returned by the IERRIST macro are listed in Table 4-2. If no error occurs, execution continues at the instruction following the IERRIST macro.

If the ERROR parameter is specified with an intercept macro and an error occurs, execution branches to the specified error routine within the intercepting task. If no error occurs, execution proceeds to the next executable source statement. The error routine pointed to by the ERROR parameter can contain an IERRIST macro to identify what error has occurred.

TABLE 4-2 ERROR CODES RETURNED FOR INTERCEPT MACROS

ERROR CODE	MEANING	RELEVANT MACROS
MO	Invalid interception mode	ICREATE
AD	Invalid address in parameter control block (PCB)	ICREATE ITERM ICONT IREMOVE ITRAP IGET IPUT
EX	Task or device exists when it should not, or does not exist when it should	ICREATE
SP	Insufficient system space to do request, or NINTC>64 or RESIZE>998	ICREATE ITERM ITRAP IGET IPUT
CT	Full control already selected	ICREATE IPOLL IPROCEED ITRAP IGET IPUT
HA	Invalid queue handler name	ICREATE
FL	Invalid device name or task name	ICREATE
ST	Invalid state for call; e.g., IPOLL followed by ICONT or issuing IPUT with monitor control intercept path	IREMOVE IPOLL IPROCEED ITRAP IGET IPUT
TF	Task queue item not added	ITRAP
RD	Invalid RDB	ITERM ICONT IPOLL IPROCEED ITRAP IGET IPUT
ID	Intercept path corresponding to	IREMOVE

	this path ID does not exist	
WP	Attempt to copy SVC parameter block back into write protected area	ITERM
CD	Invalid subcode in SVC parameter block	ALL
NT	Intercepted task has gone to end of task	IROLL IPROCEED ITRAP IGET IPUT

4.13 MACROS USED WITH SVC INTERCEPTION

Once configured for SVC interception, the operating system allows tasks to issue macros for SVC interception provided they were linked with the intercept option.

This section gives the syntax for the SVC macros described in the previous sections. Refer to the OS/32 System Macro Library Reference Manual for a list of syntax rules.

4.13.1 ICREATE Macro

The ICREATE macro creates an intercept path for a particular SVC type. See Table 4-3 for valid combinations for the SVC, MODE, and NAME parameters.

Format:

```
          |               |
   NAME   |   OPERATION   |                 OPERAND
------------------------------------------------------------------
          |               |         (1)
          |               |         (2,7)
 symbol   |   ICREATE     |  SVC=   (3)
          |               |         (6)
          |               |         (7)
          |               |
          |               |            CI
          |               |
          |               | ,MODE=     RX
          |               |
          |               |            RN
          |               |
          |               | ,NAME=pointer
```

```
,TID=pointer

          FC
,CONTROL=
          NC

,BUFFERL=pointer

,HANDLER=pointer

,PID=pointer

,EXEC=pointer

,PBSIZE=integer

[,SVAB=pointer]

[,ERROR=pointer]

[,PCB=pointer]

[,FORM=L]

[,NINTC=n]
```

Parameters:

SVC= is an integer, enclosed by parentheses, that indicates the type of intercept path to be created:

- (1) indicates SVC 1
- (2,7) indicates SVC 2 code 7
- (3) indicates SVC 3
- (6) indicates SVC 6
- (7) indicates SVC 7

MODE= indicates one of the following interception modes:

- CI indicates caller mode
- RX indicates recipient existent mode
- RN indicates recipient nonexistent mode

When CL is specified, an intercept path is created for all SVCs (selected by the SVC parameter) issued from the task specified in the NAME or TID parameter.

When RX is specified, an intercept path is created for all SVCs (selected by the SVC parameter) directed to an existing task, device, pseudo task, or pseudo device specified in the NAME parameter.

When RN is specified, a pseudo device is created for SVC 1 or SVC 7, or a pseudo task is created for SVC 6. The pseudo device or task is attached to the intercept path created by the call.

A pseudo task or pseudo device is deleted when all intercept paths attached to it are removed. When a pseudo device is assigned without SVC 7 interception, the requested access privileges are ignored and shared read/shared write privileges are granted. If an SVC 1 is attempted to a pseudo device without an interception in effect, an invalid function error (Y'C0') is returned.

NAME= indicates the address of the memory location specifying the name of a device, task, pseudo device, or task. This location must be fullword boundary aligned and contain eight bytes of blanks followed by a standard file descriptor (fd) or taskid. An fd must be packed, left-justified, and padded with blanks within the fullword. A taskid must be left-justified and padded with blanks.

When RX or RN is specified by the MODE parameter, the standard fd or taskid given with the NAME parameter can include an asterisk or a backward slash to allow generic naming. See Section 4.6.

TABLE 4-3 VALID COMBINATIONS FOR SUPERVISOR CALL (SVC), MODE, AND NAME PARAMETERS

ICREATE PARAMETERS			
SVC=	MODE=	NAME=	FUNCTION
(1)	CI	taskid	Intercepts any SVC 1 issued from the task

	RX	fd	Intercepts any SVC 1 directed to the existing device
	RN	fd	Creates a pseudo device and intercepts any SVC 1 directed to it
(2,7)	CI	taskid	Intercepts any SVC 2 code 7 issued from the task
	RX	--	No function; specifying fd or taskid results in error
	RN	--	Results in error
(3)	CI	taskid	End of task interception; occurs no matter how a task terminates
	RX	--	No function; specifying fd or taskid results in error
	RN	--	Results in error
(6)	CI	taskid	Intercepts any SVC 6 issued from the task
	RX	taskid	Intercepts any SVC 6 directed to the existing task
	RN	taskid	Creates a pseudo task and intercepts any SVC 6 directed to it
(7)	CI	taskid	Intercepts any SVC 7 issued from the task
	RX	fd	Intercepts any SVC 7 directed to the existing device
	RN	fd	Creates a pseudo device and intercepts any SVC 7 directed to it

TID= indicates the address of a fullword location containing a task identifier. This parameter, which is mutually exclusive with the NAME= parameter, can be used when MODE=CI, or MODE=RX with SVC 6, to identify the task to be intercepted. The TID can be obtained from the RDB.TID field of an RDB from a previously intercepted SVC call.

CONTROL= contains a mnemonic indicating either full control (FC) or monitor control (MC) over intercepted SVCs.

When CONTROL=FC, an intercepting task can exert full control over an application task's intercepted SVCs.

When CONTROL=MC, an intercepting task acts as a monitor only; it has no control over an intercepted SVC.

BUFFERL= indicates the address of the standard circular list that contains the addresses of available RDB buffers.

The RDB used by the intercepting task to identify an intercepted SVC must not be moved to a new location after the interception takes place. The system ensures that the address of this RDB is the same as the address of the RDB that was passed to the intercepting task when the interception occurred.

HANDLER= indicates the address of a fullword location containing the name of a queue handler. This name, a maximum of eight characters, is left-justified and padded with blanks. If this parameter is omitted, the default queue handler is invoked.

NOTE

Currently, user defined queue handlers are not supported.

PID= indicates the address of a halfword location that is used by the system to store the path identifier for the intercept path.

EXEC= is the address of an SVC intercept executor routine within the intercepting task. This routine will process intercepted SVCs of the type specified with the SVC parameter. During SVC interception, the system removes an RDB specified by the list, fills it with information, and queues a task event trap with the specified executor address to the intercepting task.

On entry to an executor routine, R0 contains the PID of the intercept path, and R1 contains the address of the RDB buffer associated with the intercepted SVC. The executor routine executes as task event service routine.

PBSIZE= is a decimal number specifying the number of bytes in the parameter block for the SVC indicated by the SVC parameter.

When this parameter is omitted, the parameter block size defaults to the standard sizes documented for each type of SVC in the OS/32 Supervisor Call (SVC) Reference Manual, except for SVC 2 code 7 interception, which defaults to eight bytes.

The size of the RDB.PB field in the RDB for this interception path is the value of the PBSIZE parameter, or its default if not specified.

SVAR= is the address of a fullword location containing user defined data. This data is passed to the intercept logic. The queue handler named by the HANDLER parameter can later access the data. The SVAR parameter is for user defined purposes when needed by a user defined queue handler.

NOTE

Currently, user defined queue handlers are not supported.

ERROR= is the address of an error routine within the intercepting task. If a run time error occurs for this macro, execution branches to this error routine.

PCB= is the address of a PCB previously constructed and initialized by the FORM=I parameter.

When no PCB parameter is included, macro code automatically builds a new PCB and initializes it with values corresponding to the other specified parameters.

FORM= I requests a PCB to be built but not executed. Macro code constructs a PCB for this macro and initializes it with values. Subsequent macros can reference this PCB via the PCB parameter.

NINTC= n specifies the number of interceptions that can execute concurrently for this intercept path. If there are more SVC interceptions outstanding than can be concurrently executed, the excess interceptions are queued. The default value for n is 1.

4.13.2 IREMOVE Macro

The IREMOVE macro allows an intercepting task to remove one or all previously created SVC intercept paths.

Format:

```
          |                |
  NAME    |   OPERATION    |              OPERAND
-----------------------------------------------------------------
symbol    | IREMOVE        |  PID=pointer
          |                |
          |                |         CS
          |                |  ,TERM=
          |                |         AB
          |                |
          |                | [,ERROR=pointer]
          |                |
          |                | [,PCB=pointer]
          |                |
          |                | [,FORM=L]
          |                |
```

Parameters:

PID= is the address of the path identifier specifying the path being removed. A zero value in the PID halfword removes all existing intercept paths.

TERM= indicates either of two termination modes for intercepted SVCs already queued for the intercepting task:

- AB indicates abort. OS/32 aborts all currently queued requests before path removal.
- CS indicates controlled shutdown. OS/32 services only currently queued requests before path removal; no requests made after TERM=CS is issued can be queued or processed.

  If this parameter is omitted, AB is the default.

ERROR= is the address of an error routine within the intercepting task. If a run time error occurs for this macro, execution branches to this error routine. If this parameter is omitted and a run time error occurs, execution resumes with the instruction following the macro.

PCB= is the address of a PCB previously constructed and initialized by the FORM=I parameter.

	If this parameter is omitted, a new PCB is automatically built and initialized with values corresponding to the other specified parameters.
FORM=I	I requests a PCB be built but not executed. A PCB is built for this macro and initialized with values. Subsequent macros can reference this PCB via the PCB parameter.

4.13.3 IGET Macro

The IGET macro allows an intercepting task to get data from the application task whose SVC is intercepted.

Format:

NAME	OPERATION	OPERAND
symbol	IGET	RDB=pointer
		,ADST=pointer
		,ADEND=pointer
		,SDST=pointer
		,SDEND=pointer
		[,ERROR=pointer]
		[,PCB=pointer]
		[,FORM=I]
		[,DONE=addr]

Parameters:

RDB=	is the address of the RDB buffer built for the intercepted SVC.
ADST=	is the start address of a data area within the application task whose SVC is intercepted. The contents of this area are transferred to an intercepting task data area.
ADEND=	is the end address of the data area within the application task whose SVC is intercepted.

SDST=	is the start address of a data area within the intercepting task. This area receives the data from the application task.
SDEND=	is the end address of the data area within the intercepting task.
ERRCR=	is the address of an error routine within the intercepting task. If a run time error occurs for this macro, execution branches to this error routine. If this parameter is omitted and a run time error occurs, execution resumes with the instruction following the macro.
PCB=	is the address of a PCB previously constructed and initialized by the FORM=I parameter. If this parameter is omitted, a new PCB is built and initialized with values corresponding to the other specified parameters.
FORM=	I requests a PCB be built but not executed. A PCB is built for this macro and initialized with values. Subsequent macros can reference this PCB via the PCB parameter.
DONE=	is an address that specifies that the macro is to be a proceed call. When the call is completed, a task event interrupt occurs, using the routine specified by the address in the DONE parameter. This routine enters with R0 containing the error code for the call and R1 pointing to the macro's parameter block. Once this routine has finished processing, it exits using the TEXIT macro. The proceed form of the IGET macro must be used if an IRCLL macro was issued to the application task whose SVC is intercepted. The system cannot guarantee that the application task is in memory or that it can be rolled into memory within a reasonable time.

4.13.4 IPUT Macro

The IPUT macro lets an intercepting task put data into a data area of the application task whose SVC is intercepted.

Format:

NAME	OPERATION	OPERAND
symbol	IPUT	RDB=pointer
		,ADST=pointer
		,ADEND=pointer
		,SDST=pointer
		,SDEND=pointer
		[,ERROR=pointer]
		[,PCB=pointer]
		[,FORM=I]
		[,DONE=addr]

Parameters:

RDB= is the address of the RDB buffer built for the intercepted SVC.

ADST= is the start address of a data area within the application task. This area receives the contents of an intercepting task data area.

ADEND= is the end address of the data area within the application task.

SDST= is the start address of a data area within the intercepting task. The contents of this area are put into the application task data area.

SDEND= is the end address of the data within the application task.

ERROR= is the address of an error routine within the intercepting task. If a run time error occurs for this macro, execution branches to this error routine.

If this parameter is omitted and a run time error occurs, execution resumes with the instruction following the macro.

PCB= is the address of a PCB previously constructed and initialized by the FORM=I parameter. If this parameter is omitted, a new PCB is automatically built and initialized with values corresponding to the other specified parameters.

FORM= L requests a PCB be built but not executed. A PCB is built for this macro and initialized with values. Subsequent macros can reference this PCB via the PCB parameter.

DONE= is an address that specifies that the macro is to be a proceed call. When the call is completed, a task event interrupt occurs, using the routine whose address is specified in the DONE parameter. This routine enters with R0 containing the error code for the call, and R1 pointing to the macro's parameter block. Once this routine has finished processing, it exits using the TEXIT macro.

The proceed form of the IPUT macro must be used if an IROLL macro was issued to the application task. The system cannot guarantee that the application task is in memory or that it can be rolled into memory within a reasonable time.

4.13.5 ICONT Macro

The ICONT macro returns control of an intercepted SVC by returning control to an OS/32 SVC executor.

Format:

NAME	OPERATION	OPERAND
symbol	ICONT	RDB=pointer
		[,ERROR=pointer]
		[,PCB=pointer]
		[,FORM=L]

Parameters:

RDB= is the address of the RDB buffer built for the intercepted SVC.

ERROR= is the address of an error routine within the intercepting task. If a run time error occurs for this macro, execution branches to this error routine.

If this parameter is omitted and a run time error occurs, execution resumes with the instruction following the macro.

PCB= is the address of a PCB previously constructed and initialized by the FORM=I parameter.

If this parameter is omitted, a new PCB is automatically built and initialized with values corresponding to the other specified parameters.

FORM= I requests a PCB be built but not executed. A PCB is built for this macro and initialized with values. Subsequent macros can reference this PCB via the PCB parameter.

4.13.6 IPROCEED Macro

After an SVC has been intercepted, the intercepting task can issue an IPROCEED macro to allow the application task that issued the SVC to proceed with its execution. Until the intercepting task issues an IPROCEED macro, the application task is in wait state.

Format:

NAME	OPERATION	OPERAND
symbol	IPROCEED	RDB=pointer
		[,ERROR=pointer]
		[,PCB=pointer]
		[,FORM=L]
		[,CC=n]

Parameters:

RDB= is the address of the RDB buffer built for the intercepted SVC.

ERROR= is the address of an error routine within the intercepting task. If a run time error occurs

	for this macro, execution branches to this error routine. If this parameter is omitted and a run time error occurs, execution resumes with the instruction following the macro.
PCB=	is the address of a PCB previously constructed and initialized by the FORM=I parameter. If this parameter is omitted, a new PCB is automatically built and initialized with values corresponding to the other specified parameters.
FORM=	I requests a PCB be built but not executed. A PCB is built for this macro and initialized with values. Subsequent macros can reference this PCB via the PCB parameter.
CC=	n is a decimal number specifying the setting of the application task PSW condition code after the SVC instruction execution. If the CC parameter is omitted, the condition code of the application task PSW is set to zero.

4.13.7 IROLL Macro

After an SVC is intercepted an IROLL macro lets an intercepting task change the status of the application task from nonrollable to rollable, provided that the task was established as rollable by Link. This allows OS/32 to roll out a task whose intercepted SVC requires lengthy processing.

Unless an IROLL macro is issued, an application task cannot be rolled after its SVC is intercepted, even if an IPROCEED macro is issued. However, an IROLL macro can be issued after an IPROCEED macro is issued.

Format:

NAME	OPERATION	OPERAND
symbol	IROLL	RDP=pointer
		[,ERROR=pointer]
		[,PCB=pointer]
		[,FORM=L]

Parameters:

RDB= is the address of the RDB buffer built for the intercepted SVC.

ERROR= is the address of an error routine within the intercepting task. If a run time error occurs for this macro, execution branches to this error routine. If this parameter is omitted and a run time error occurs, execution resumes with the instruction following the macro.

PCB= is the address of the PCB previously constructed and initialized by the FORM=L parameter. If this parameter is omitted, a new PCB is automatically built and initialized with values corresponding to the other specified parameters.

FORM= L requests a PCB be built but not executed. A PCB is built for this macro and initialized with values. Subsequent macros can reference this PCB via the PCB parameter.

4.13.8 ITERM Macro

The ITERM macro allows an intercepting task to return the parameter block of the SVC it processed to the application task that issued the SVC. The returned parameter block can have updated information such as status, number of bytes transferred, etc.

Format:

```
               |               |
    NAME       |   OPERATION   |              OPERAND
---------------------------------------------------------------------
  symbol       | ITERM         |  RDB=pointer
               |               |
               |               | [,TPAP=pointer]
               |               |
               |               |          Y
               |               |  ,COPY=
               |               |          N
               |               |
               |               | [,ERROR=pointer]
               |               |
               |               | [,PCB=pointer]
               |               |
               |               | [,FORM=L]
               |               |
               |               | [,CC=n]
               |               |
```

Parameters:

RDB=	is the address of the RDB buffer built for the intercepted SVC.
TRAP=	is the address of a fullword to be added to the task queue of the application task whose SVC is intercepted.
COPY=	Y (yes) indicates that the SVC parameter block in the RDB is to be copied back into the parameter block of the intercepted SVC. N (no) indicates the operation is not performed. If this parameter is omitted, N is the default.
ERROR=	is the address of an error routine within the application task whose SVC is intercepted. If a run time error occurs for this macro, execution branches to this error routine. If this parameter is omitted and a run time error occurs, execution resumes with the instruction following the macro.
PCB=	is the address of a PCB previously constructed and initialized by the FORM=I parameter. If this parameter is omitted, a new PCB is automatically built and initialized with values corresponding to the other specified parameters.
FORM=	I requests a PCB be built but not executed. A PCB is built for this macro and initializes it with values. Subsequent macros can reference this PCB via the PCB parameter.
CC=	n is a decimal number specifying the setting of the application task PSW condition code after the SVC instruction execution. If the CC parameter is omitted, the condition code of the application task PSW is set to zero.

4.13.9 ITRAP Macro

The ITRAP macro allows an intercepting task to send a task queue item to an application task whose SVC is intercepted. The task queue item can be any of the task queue items supported by CS/32.

Format:

NAME	OPERATION	OPERAND
symbol	ITRAP	RDB=pointer
		TID=pointer
		TRAP=pointer
		[,ERROR=pointer]
		[,PCB=pointer]
		[,FORM=I]
		[,DONE=addr]

Parameters:

RDB= is the address of the RDB buffer built for the intercepted SVC.

TID= is the address of a fullword containing the taskid for the task. Before issuing an ITRAP macro with the TID parameter, the intercepting task must have obtained the task identifier from and RDB and placed it into the fullword location.

NOTE

The TID form of this macro can be used to send a trap to a task that is not being intercepted.

TRAP= is the address of a fullword to be added to the task queue of the application task whose SVC is intercepted.

ERROR= is the address of an error routine within the intercepting task. If a run time error occurs for this macro, execution branches to this error routine. If this parameter is omitted and a run time error occurs, execution resumes with the instruction following the macro.

PCB= is the address of a PCB previously constructed and initialized by the FORM=I parameter. If this parameter is omitted, a new PCB is automatically built and initialized with values corresponding to the other specified parameters.

FORM= L requests a PCB be built but not executed. A PCB is built for this macro and initializes it with values. Subsequent macros can reference this PCB via the PCB parameter.

DONE= is a hexadecimal number specifying the address of the task event service routine. When an I/O proceed call is completed, a task event interrupt occurs, using the routine whose address is specified in the DONE parameter. This routine enters with R0 containing the error code for the call and R1 pointing to the macro's parameter block. Once this routine has finished processing, the intercepting task exits from the task event trap using the TEXIT macro.

The proceed form of the ITRAP macro must be used if an IPCLL macro was issued to the application task whose SVC is intercepted. The system cannot guarantee that the application task is in memory or that it can be rolled into memory within a reasonable time.

4.13.10 IERRTST Macro

The IERRTST macro allows an intercepting task to evaluate errors resulting from intercept macros in order to branch to appropriate error handling routines.

Format:

NAME	OPERATION	OPERAND
symbol	IERRTST	xx=pointer
		.
		.
		.
		xx=pointer
		ELSE=pointer
		PCB=pointer
		FORM=L

Parameters:

xx= — is a two-character alphabetic string specifying one of the error codes for the intercept macros. See Table 4-2.

Pointer — specifies the address of an intercepting task error routine that handles errors having a returned error code identical to the one specified by the xx parameter. For instance, an IERRTST macro might include these parameters for evaluating an IPUT macro:

```
IERRTST   AD=address,NT=address,RD=address
```

These four parameters specify error routine addresses to branch to whenever the returned error code equals AD, NT, or RD.

ELSE= — is the address of an error routine to be executed for errors other than those specified in the xx parameter. If this parameter is omitted, either of the following actions occurs for returned errors:

- If the returned error code corresponds to the one specified by the xx parameter, execution branches to a specific error routine.
- If the returned error code does not correspond to the one specified by the xx parameter, execution branches to the instruction immediately following the IERRTST macro.

PCB= — is the address of a PCB previously constructed and initialized by the FORM=I parameter. If this parameter is omitted, a new PCB is automatically built and initialized with values corresponding to the other specified parameters.

FORM=I — I requests a PCB be built but not executed. A PCB is built for this macro and initialized with values. Subsequent macros can reference this PCB via the PCB parameter.

4.13.11 $RDB Macro

The $RDB macro is used to define a structure containing the symbolic names for all of the RDB fields. It is recommended that symbolic names be used to reference the RDB fields instead of coding the hexadecimal offsets to the fields.

Format:

NAME	OPERATION	OPERAND
[symbol]	$RDB	

4.14 SAMPLE SVC INTERCEPTION PROGRAM

The following program uses SVC interception software to intercept SVC 1 to the device MAG1. Each time SVC 1 is issued, the program prints out "SVC 1 call intercepted". The SVC 1 is terminated with a device unavailable error code (Y'A0').

```
    >$RDB                       /DEFINES AN RDB STRUCTURE

*Add an RDB buffer address to the RDB buffer address list.

    >LA      0,RDB              /LOAD THE ADDRESS OF THE RDB
                                 INTO REGISTER 0

    >ABL     0,BUFLIST          /ADD THE ADDRESS OF THE RDB
                                 TO THE CIRCULAR LIST

*Create the Intercept Path

    >ICREATE NAME=INTNAME,      /SPECIFIES FD FOR DEVICE NAME

    >MODE=RX,                   /SPECIFIES RESIDENT-EXISTENT MODE

    >CONTROL=FC,                /GIVES INTERCEPTING TASK FULL CONTROL
                                 OVER INTERCEPTED SVC

    >SVC=(1),                   /SPECIFIES THAT ALL SVC 1 TYPES ARE
                                 TO BE INTERCEPTED

    >EXEC=INTRTN,               /POINTS TO THE SVC EXECUTOR ROUTINE
                                 THAT SERVICES THE INTERCEPTED SVC

    >BUFFERL=BUFLIST,           /ASSIGNS POINTER TO CIRCULAR LIST
                                 CONTAINING ADDRESSES OF FREE
                                 RDB BUFFERS

    >PID=PATHID,                /DEFINES DATA AREA FOR INTERCEPT PATH
                                 IDENTIFIER

    >ERROR=BOMBOUT              /DESIGNATES ERROR ROUTINE TO WHICH THE
                                 PROGRAM WILL BRANCH TO IF RUN-TIME
```

```
                        ERROR OCCURS IN ICREATE MACRO

*Enable task event  trap so task can go into trap wait for inter-
 cepts to occur

    >LTSW                   /LOAD TSW WITH WAIT STATE SET

    >TEIS,WI                /TASK EVENT TRAPS ENABLED

>Come here if error occurs in ICREATE macro

    >BOMBOUT    SVC 3,1     /FAIL TASK ON ERROR

>Allocate data area for ICREATE

    >           ALIGN 4
    >INTNAME    DC    C'    '         /NODE NAME
    >           DC    C'MAG1'         /DEVICE NAME
    >           DC    C'    '         /FILE NAME
    >           DC    C'    '         /EXTENSION

    >BFULIST  DLIST 1       /DESIGNATE AREA FOR 1 RDB IN CIRCULAR
                             LIST

    >RDB   DS RDB.PB+20     /ALLOCATES SIZE OF RDB + SVC 1 PARAMETER
                             BLOCK

    >PATHID  DS2            /DESIGNATE AREA FOR ICREATE MACRO TO PUT
                             THE PATH ID

*TRAP EVENT SERVICE ROUTINE
*THE FOLLOWING ROUTINE IS EXECUTED WHEN AN SVC IS INTERCEPTED

    >INTRTN  SVC 2,NOTIFY   /LOG MESSAGE THAT AN SVC WAS INTER-
                             CEPTED

             LHI  0,X'AC00' /GET STATUS FOR INTERCEPTED SVC 1

             STH  0,RDB.PB+2(1)  /SAVE SVC 1 STATUS IN STATUS FIELD
                                  OF RDB

*TERMINATE THE INTERCEPTED CALL, COPYING THE MODIFIED SVC
*PARAMETER BLOCK IN THE RDB BACK OVER THE USER'S SVC PARAMETER
*BLOCK.

    >ITERM RDB=(1),COPY=Y

    >TEXIT              /EXIT THE TASK EVENT ROUTINE

*ALLOCATE DATA AREA FOR EVENT SERVICE ROUTINE
>       ALIGN 4
>       NOTIFY DB   0,7,0,22
>       DC   C'SVC 1 CALL INTERCEPTED'
>       END
```

CHAPTER 5
OS/32 SUPPORTED I/O DEVICES AND DEVICE DEPENDENT AND INDEPENDENT INFORMATION

5.1 INTRODUCTION

All I/O requests are made via I/O macros. This chapter discusses the functional aspects of the devices supported by OS/32. Specific device dependent information (supported functions, status returned, and formatting performed) is included.

OS/32 devices and files support ASCII formatting, proceed I/O, sequential access, unconditional and conditional proceed, and uniform vertical forms control (VFC). Device codes, ranging from 0 through 255, are associated with Perkin-Elmer supported devices and are listed in the OS/32 System Generation (SYSGEN) Reference Manual.

5.2 UNIFORM VFC

VFC support allows all OS/32 ASCII output devices to comply with American National Standards Institute (ANSI) forms control standards for FORTRAN and COBOL programming.

ANSI requires that the logical vertical position of a form always match the physical position of the form when it is sent to an output device, including interactive devices. In addition, interactive devices must be able to intermix the following operations in a consistent manner:

- o read with VFC,
- o read without VFC,
- o write with VFC, and
- o write without VFC.

Perkin-Elmer VFC support complies with the FORTRAN carriage control character set defined by ANSI X3.9-1978 for forms control before printing. If an I/O device is used that does not support a certain VFC character, inputting that character will activate a single line feed before printing.

5.3 MIXED VFC AND NON-VFC OPERATIONS

VFC and non-VFC I/O operations perform similarly. Read operations alternating with write operations, which have single line spacing (before or after), are on alternating lines.

Read with and without VFC operations start at the current physical cursor position and end with a carriage return (CR) and single line feed (LF). The relationship of logical and physical cursor position is the same before and after a read operation is performed.

The following examples describe mixed VFC and non-VFC I/O operations. Each example is based on the previous one. Initially, the logical and physical positions of the cursor are set to line 1. Note the changes in logical and physical cursor positions and line numbers as each I/O operation is performed.

The abbreviations used in the examples are as follows:

ABBREVIATION	MEANING
CL	Current logical position
PL	Previous logical position
CP	Current physical position
PP	Previous physical position
CR	Carriage return
LF	Line feed

Example 1: Write with one line space before printing

After spacing, CL and CP move to line 2, and PP and PL are situated at line 1. The characters that result from this operation are printed on line 2. After printing, a CR/LF is performed, moving CP to line 3. The resulting cursor positions are:

CURSOR POSITIONS	LINE NUMBER	CHARACTERS PRINTED
PL, PP	1	
CL	2	EXAMPLE 1 PRINT
CP	3	

Example 2: Write with one line space before printing

After spacing, PL moves to line 2 and PP to line 3. When PL and PP are not located on the same line, PP must point back to PL for the I/O operation to be performed. The characters that result from this operation are printed on line 3. Both CL and CP are then moved to line 3. After printing, a CP/LF is performed, moving CP to line 3. The resulting cursor positions are:

CURSOR POSITIONS	LINE NUMBER	CHARACTERS PRINTED
PL	2	EXAMPLE 1 PRINT
PP, CL	3	EXAMPLE 2 PRINT
CP	4	

Example 3: Write with one line space after printing

Before printing, PP points to line 3 as shown in Example 2. The characters printed as a result of the I/O operation are printed on line 3, which overprints the output from Example 2. After moving one line space, CL and CP are positioned at line 4. The resulting cursor positions are:

CURSOR POSITIONS	LINE NUMBER	CHARACTERS PRINTED
PL	3	EXAMPLE 3 PRINT
PP, CL, CP	4	

Example 4: Write with one line space after printing

Before printing, PL and PP are positioned at line 4. The characters printed as a result of the I/O operation are printed on line 4. After moving one line space, CL and CP are positioned at line 5. The resulting cursor positions are:

CURSOR POSITIONS	LINE NUMBER	CHARACTERS PRINTED
PL, PP	4	EXAMPLE 4 PRINT
CL, CP	5	

Example 5: Write with no VFC

After printing, PL and PP are positioned at line 5. The characters printed as a result of the I/O operation are printed on line 5. After CR/LF, CL and CP are positioned at line 6. The resulting cursor positions are:

CURSOR POSITIONS	LINE NUMBER	CHARACTERS PRINTED
PL, PP	5	EXAMPLE 5 PRINT
CL, CP	6	

Example 6: Write with one line space before printing

After spacing one line, PL and PP are positioned at line 6 and CL and CP are positioned at line 7. The characters printed as a result of the I/O operation are printed on line 7. After CR/LF, CP is positioned at line 8, while CL remains at line 7. The resulting cursor positions are:

CURSOR POSITIONS	LINE NUMBER	CHARACTERS PRINTED
PL, PP	6	
CL	7	EXAMPLE 6 PRINT
CP	8	

Example 7: Write with no VFC

Before printing, PP points back to line 7. The characters printed as a result of the I/O operation are printed on line 7, which overprints the output from Example 6. PP, CL and CP are moved to line 8. The resulting cursor positions are:

CURSOR POSITIONS	LINE NUMBER	CHARACTERS PRINTED
PL	7	EXAMPLE 7 PRINT
PP, CL, CP	8	

Example 8: Read with VFC

Before characters are read with VFC, PL and PP are positioned at line 8. As a result of the I/O operation the characters are read from line 8. CL and CP are moved to line 9. The resulting cursor positions are:

CURSOR POSITIONS	LINE NUMBER	CHARACTERS READ
PL, PP	8	EXAMPLE 8 READ
CL, CP	9	

Example 9: Read without VFC

Before characters are read without VFC, PL and PP are positioned at line 9. As a result of the I/O operation the characters are read from line 9. CL and CP are moved to line 10. The resulting positions are:

CURSOR POSITIONS	LINE NUMBER	CHARACTERS READ
PL, PP	9	EXAMPLE 9 READ
CL, CP	10	

Example 10: Write with one line space before printing

Before spacing one line, PL and PP are positioned at line 10. CL and CP are positioned at line 11 after moving one line space. The characters printed as a result of the I/O operation are printed on line 11. After CR/LF, CP is positioned at line 12. The resulting cursor positions are:

CURSOR POSITIONS	LINE NUMBER	CHARACTERS PRINTED
PL, PP	10	
CL	11	EXAMPLE 10 PRINT
CP	12	

Example 11: Read with VFC

Before reading with VFC, PP points back to line 11. As a result of the I/O operation the characters are read from line 12. After CR/LF, CL is positioned at line 12, and CP is positioned at line 13. The resulting cursor positions are:

CURSOR	LINE	CHARACTERS PRINTED

POSITIONS	NUMBER	AND READ
PL	11	EXAMPLE 10 PRINT
PP, CL	12	EXAMPLE 11 READ
CP	13	

Example 12: Write with one line space before printing

Before spacing, PP points back to line 12. After moving one line space, CL and CP are positioned at line 13. The characters printed as a result of the I/O operation are printed on line 13. After CR/LF, CP is positioned at line 14, while CL remains at line 13. The resulting cursor positions are:

CURSOR POSITIONS	LINE NUMBER	CHARACTERS PRINTED
	11	EXAMPLE 10 PRINT
PL	12	
PP, CL	13	EXAMPLE 12 PRINT
CP	14	

5.4 CARD EQUIPMENT

Perkin-Elmer card readers can accommodate a fixed record length of 80 bytes (ASCII), 120 bytes (binary), or 160 bytes (image).

During read ASCII operations, each card column (12 bits) is converted into one 8-bit ASCII character. Illegal codes are converted into the null character (X'00') indicating an error has occurred.

During read binary operations, each pair of card columns (12 bits each) is unpacked into three bytes having the following format:

First card column												Second card column									
11	10	0	1	2	3	4	5	6	7	8	9	11	10	0	1	2	3	4	5	6	7

Bytes:
0 (first byte), 1, 2

During read image operations, each column is converted into one halfword in the following format (U=undefined):

```
 -----------------------------------------------
| U| U|11|10| 0| 1| 2| 3| U| U| 4| 5| 6| 7| 8| 9|
 -----------------------------------------------
Bytes:
0                                             15
```

STATUS	MEANING
X'C0'	Normal end of transfer
X'A0'	Device unavailable; reader not ready
X'82'	Hopper empty or stacker full
X'84'	Data transfer error (read check or pick check)
X'C0'	Illegal function

The translation for an ASCII read is accomplished through a translation table. The devices without hardware translation translate 029- or 026-compatible Hollerith code to 8-bit ASCII code. Source sysgen options include translation of 029- or 026-compatible Hollerith code to EBCDIC code. The hardware translation matches that of the 029-compatible Hollerith translation.

Card reader/punch devices supported by Perkin-Elmer 32-bit computers accommodate fixed record lengths of 80 bytes (ASCII), 120 bytes (column binary), and 160 bytes (image).

During read ASCII operations, each card column (12 bits) is converted into one 8-bit ASCII character. Illegal codes are converted into the null character (X'00') indicating an error has occurred.

During read binary operations, each pair of card columns (12 bits each) is unpacked into three bytes having the following format:

```
           First card column                        Second card column

 -----------------------------------------------------------------------------------------------
|11|10| 0| 1| 2| 3| 4| 5| 6| 7| 8| 9|11|10| 0| 1| 2| 3| 4| 5| 6| 7| 8| 9
 -----------------------------------------------------------------------------------------------
Bytes:
0                        1                        2                                             3
```

During read image operations, each card column (12 bits each) is placed into a halfword in the following format:

```
 -----------------------------------------------
| 0| 0| 0| 0|11|10| 0| 1| 2| 3| 4| 5| 6| 7| 8| 9|
 -----------------------------------------------
Bits:
0                                             15
```

During write ASCII operations, each byte of data is translated from ASCII into a 12-bit Hollerith code. Depending on the device code chosen, the following can occur:

- o All data is punched and printed.
- o Data is punched only.
- o Of each 160 bytes of data accepted, the first 80 bytes are punched while the second 80 bytes are printed.

During write binary operations, each 3-byte group is packed into two columns on the card in the following format. Nothing is printed on top of the card.

Odd column	0	1	2	3	4	5	6	7	0	1	2	3
Even column	4	5	6	7	0	1	2	3	4	5	6	7

During write image operations, the low order 12 bits of each halfword are punched according to the following format. Nothing is printed on top of the card. Bits 0 through 3 are ignored.

				12	11	0	1	2	3	4	5	6	7	8	9

Bits:
0 15

STATUS	MEANING
X'C0'	Normal end of transfer
X'A0'	Device unavailable
X'82'	Hopper empty, stacker full
X'84'	Data transfer error
X'C0'	Illegal function

The translation for ASCII operations is accomplished through a translation table. The standard translation is 8-bit ASCII code to 029-compatible Hollerith code.

Source sysgen options include 8-bit ASCII code to 026-compatible Hollerith code and also EBCDIC code to 026- or 029-compatible Hollerith code.

5.5 TELETYPE (TTY) READER/PUNCH

Perkin-Elmer TTY reader/punch devices support read and write ASCII, read and write binary, and read and write image operations. Variable length records are also accommodated.

During read ASCII operations, an X-ON character is output to turn the reader on. The tape is read in blocked mode so data is not printed on the printer while it is being read. Leading blank frames and delete characters are ignored. Data is masked to 7-bit ASCII. The transfer is terminated on buffer full or carriage return, whichever occurs first. On termination of the transfer, the tape is advanced to the next delete character or blank frame. An X-OFF character is output to stop the tape.

During binary read operations, an X-ON character is output to turn on the tape. The tape is skipped until the first nonblank frame is found. If the first nonblank character read is an X'F0', the following frames are read in until the user buffer is full. The characters read in are in unzoned binary format. If the first nonblank character read is not an X'F0', zoned binary mode is assumed. In this case, the characters are read, stripped of the zones, and packed into the user buffer until the buffer is full. In this mode, the only valid punches are X'90', X'81' through X'84', and X'95' through X'9F'. Other characters are ignored. On buffer full, the tape is advanced to the next blank frame in zoned binary mode. In unzoned binary, the first character transferred to the user buffer is the character following the X'F0' character; while in zoned binary, the first nonblank frame is transferred (after stripping and packing).

During read image operations, none of the above formatting operations are performed. An X-ON character is output to turn the tape on, and data is read until the user buffer is full. The X-OFF character is then output to turn the tape off and the transfer is complete.

During write ASCII operations, the driver outputs a RUBOUT-TAPE RUBOUT-RUBOUT sequence in order to initialize the TTY reperforator. Eight frames of blank tape are output as leader, the user data is output until the buffer is empty, or carriage return, whichever occurs first. The driver ensures that a CR-LF-TAPE OFF-RUBOUT sequence terminates the record.

During write binary operations, the driver outputs a RUBOUT-TAPE-RUBOUT-RUBOUT sequence, followed by eight blank frames of leader. The user buffer is output, translating each byte into two frames of zoned binary data. The transfer is terminated when the buffer is empty. The driver outputs a TAPE OFF-RUBOUT sequence.

During write image operations, none of the above formatting or control operations are performed. The user buffer is output until the buffer is empty.

STATUS	MEANING
X'00'	Normal completion
X'A0'	Device unavailable
X'84'	Data transfer error
X'82'	Break detected during transfer. Timeout.
X'C0'	Illegal function

On ASCII or image write, it is possible to inadvertently turn off the punch by outputting a TAPE OFF character. On image write, it is the responsibility of the user to place the necessary control characters, such as TAPE and TAPE OFF, in the user buffer to control the operation of the tape.

Since the reader/punch portion of the TTY is connected to the keyboard/printer portion, only one of these devices can be active at a time. On ASCII write, the data punched on the tape is also printed on the printer.

5.6 TTY KEYBOARD/PRINTER

Perkin-Elmer TTY keyboard/printers accommodate variable length records and can be interfaced to current loop devices.

In read ASCII operations, data read is masked to 7-bit ASCII. Data is read until the buffer is full or a carriage return is found, whichever occurs first. Upon termination, a carriage return/line feed (CR/LF) sequence is sent to the printer. Typing CNTRL X causes the line input to be ignored, a CR/LF sequence to be output, and the read operation to be restarted. Typing CNTRL H causes the previous character entered to be ignored.

In write ASCII operations, the buffer is scanned to eliminate trailing blanks. Data is then output until the buffer is exhausted or until a carriage return is found in the data stream. A line feed is automatically appended to the detected carriage return; or if no carriage return is detected, a CR/LF sequence is output.

During image I/O, none of the above formatting actions occur. The amount of data requested is typed out or read in, without masking to 7-bit ASCII, eliminating trailing blanks, checking for CNTRL X or CNTRL H characters, or detecting or appending carriage returns or line feeds. On image read, however, a carriage return is detected as an end of line sentinel.

STATUS	MEANING
X'00'	Normal completion
X'82'	Timeout or line break
X'84'	Unrecoverable error
X'A0'	Device unavailable

While the reader/punch of an ASR TTY is treated as a separate device, it cannot operate simultaneously with the keyboard/printer.

5.7 PAPER TAPE EQUIPMENT

Variable record lengths are supported by Perkin-Elmer paper tape devices. During read ASCII operations, leading blank tape and delete characters are ignored. Data is masked to 7-bit ASCII. Carriage return terminates read. On termination, the tape is advanced until either a blank tape or a delete character is read.

During read binary operations, tape is advanced until a nonzero character is read. If this character is X'F0', the tape is read until the buffer is full (unzoned binary). If the first nonzero character is not X'F0', the tape is treated as a zoned binary tape. Each two characters are stripped of their zone, merged into one byte, and placed in the buffer until the buffer is full. On buffer full, the tape is advanced until blank tape is found. In zoned binary mode, the only valid characters are: X'90', X'81' - X'84' and X'95' - X'9F'. All other characters cause the transfer to end with unrecoverable status.

During read image operations, the tape is read until the buffer is full.

During write ASCII operations, eight frames of blank tape are output. The user buffer is output up to (but not including) carriage return or until buffer empty. CR/LF is then output.

During write binary operations, eight frames of blank tape, followed by the character X'F0', are output. The user buffer is output until the buffer is empty.

During write image operations, the user buffer is output until the buffer is empty.

STATUS	MEANING
X'C0'	Normal completion
X'A0'	Device unavailable
X'82'	Timeout
X'84'	Transfer error or invalid zone character
X'C0'	Illegal function

5.8 LINE PRINTERS

Perkin-Elmer line printers support variable record lengths up to 132 bytes.

During write ASCII operations, the user buffer is output until a carriage return is found or until the buffer is empty. At buffer termination, the system takes all necessary action to ensure that the buffer is printed and the paper is spaced upward one line. If form-feed or other paper motion is desired, the appropriate characters must appear in the user buffer.

During write image operations, the user buffer is output exactly as in memory. The system does not take action to ensure that the data is printed or that the paper is properly moved. The user should be familiar with the characteristics of the particular device being used.

STATUS	MEANING
X'C0'	Normal completion
X'A0'	Device unavailable. Device not ready Form error
X'82'	Device timeout
X'84'	Device interlock
X'C0'	Illegal function

Although the low and high speed line printers have different form control characters, the sequence carriage return, X'01', causes a new line to be started on each printer. This sequence, used for write ASCII, allows program compatibility using the same driver.

5.9 TAPE CASSETTE

Variable length records are supported by Perkin-Elmer tape cassettes. During input, ASCII, binary, and image modes are identical. Data is read from the cassette into the user buffer. The transfer terminates when the buffer is full or at end of record, whichever comes first. If the record is longer than the buffer, error status is not returned. Parity errors in the unread part of the record can be detected. If a parity error occurs, five retries are attempted before error status is returned. When a parity error status is returned, the tape is positioned in the interrecord gap following the record in error.

During output, ASCII, binary, and image modes are identical. Data is written from the user buffer until the buffer is empty. The system retries five times on parity errors.

STATUS	MEANING
X'C0'	Normal completion
X'A0'	Device not ready; tape failed to move at start of request
X'90'	End of tape on read, write, or write file mark
X'88'	End of file
X'84'	Device became unavailable during a request
X'82'	Data transfer error after five retries;

	timeout
X'98'	End of file/end of tape
X'C0'	Illegal function

The driver generates an end of tape condition, whether the tape is positioned at the beginning or at the end of the reel. It must be assumed from the last operation what position end of tape is actually referring to.

Since the two drives on an intertape cassette share logic, only one drive of a cassette pair; e.g., X'45' and X'55', can be active at a time.

Continuous mode operations are used to pass requests to the driver within the time required (10 milliseconds for read; 30 milliseconds for backspace).

5.10 MAGNETIC TAPE

Variable length records are supported by Perkin-Elmer magnetic tape devices. During input, data is read into the user buffer from the magnetic tape. The transfer ends on buffer full or end of record, whichever comes first. If the record is longer than the user buffer, error status X'82' is returned. This error status code also indicates parity error. On a parity error, five retries are attempted before error status is returned. After a parity error, the tape is positioned in the interrecord gap following the record with the error.

During output, data is written from the user buffer to the magnetic tape until the buffer is empty. On parity error, an extended record gap is written and the write is retried.

For read and write requests, ASCII, binary, and image requests are identical.

STATUS	MEANING
X'00'	Normal completion
X'A0'	Device not ready; tape unavailable at the start of request for data transfer
X'90'	End of tape; request caused the reflective marker at the beginning or end of tape to be sensed
X'88'	End of file; filemark detected during request
X'84'	Device became unavailable during request
X'82'	Data transfer error after 5 retries; or record transferred is longer than user buffer
X'C0'	Illegal function
X'98'	End of file detected concurrently with end of tape marker

The driver assumes the tape is at end of tape if end of tape is detected on a write request. On a read operation, end of tape may be detected on a different record than on a write operation because of mechanical tape positioning. If rewind is issued at beginning of tape, the driver returns normal status. Ensure that the tape is loaded at beginning of tape unless some other condition is expected.

5.11 DISK STORAGE

Perkin-Elmer disk devices support variable length records. During input, a current sector pointer is maintained. On a sequential read, data is read into the user buffer from the disk, starting at the current sector, until the buffer is full. If an attempt is made to read beyond the end of the disk, end of medium (EOM) status is returned. On a random request, data is read from the disk starting at the sector specified by the random sector address passed with the request, until the buffer is full. If an attempt is made to read beyond the end of the disk, EOM status is returned with data transferred. ASCII, binary, and image requests are identically treated.

During output, data is written from the user buffer to the disk, starting at the current sector (for sequential writes) or at the specified sector (for random writes), until the buffer is empty. Attempts to write past the end of the disk cause EOM status to be returned. In this case, no data is transferred.

Errors on data transfers cause the operation to be retried several times before returning error status.

All data transfers start on a sector boundary, but can end on any byte of a sector.

STATUS	MEANING
X'00'	Normal completion
X'A0'	Request could not be started, device not ready
X'90'	EOM; transfer ends beyond the end of the disk
X'84'	An unrecoverable error occurred on a 2.5, 5, or 40Mb disk, and retry efforts failed
X'82'	A recoverable error was detected on a 2.5, 5, or 40Mb disk
X'83'	Write attempted to protected drive
X'FF'	Selector channel (SELCH) end address was not within the bounds of the transfer requested

If an error condition other than one of those mentioned above occurs, the device dependent status is set to the hardware status of the file controller, disk drive, or SELCH, depending upon the error condition which prevailed.

The file manager uses the moving head disk driver. A user program cannot invoke the disk driver unless it is an e-task. For u-tasks, the disk is accessed via the contiguous or indexed file handlers.

5.12 FLOPPY DISK

Variable length records are supported by Perkin-Elmer floppy disks. During input, a current sector pointer is maintained. On a sequential read, data is read from the disk starting at the current sector into the user buffer until the buffer is full. On a random request, the data is read from the disk starting at the sector specified by the random sector address passed with the request, until the buffer is full. If an attempt is made to read beyond the end of disk, EOM status is returned with data transferred. ASCII, binary, and image requests are identically treated.

During output, data is written from the user buffer to the disk, starting at the current sector pointer (for sequential writes) or at the specified sector (for random writes), until the buffer is empty. If an attempt is made to write beyond the end of the disk, EOM status is returned with no data transferred. ASCII, binary, and image requests are identically treated.

Errors on data transfers cause the operation to be retried ten times before returning error status.

All data transfers start on a logical 256-byte sector boundary (two physical sectors on the floppy). Transfer can end on any byte of a sector.

STATUS	MEANING
X'00'	Normal completion
X'A0'	Request could not be started; device not ready
X'90'	EOM; transfer ends beyond the end of data
X'C0'	Illegal function
X'84'	All device errors other than data transfer error
X'82'	Data transfer error after ten retries
	Write protection violation; timeout

The floppy disk driver is designed for use by the file manager. A user program cannot invoke the driver unless it is an e-task. For u-tasks, the disk is accessed by the contiguous or indexed file handlers.

5.13 VIDEO DISPLAY UNIT (VDU) TERMINALS

Variable length records are supported by all Perkin-Elmer VDU terminals.

During read ASCII operations, data read is masked to 7-bit ASCII. Data is read until the buffer is full or a carriage return is found, whichever occurs first. Upon termination, a CR/LF sequence is sent to the screen. CNTRL X causes the line input to be ignored, an LF/CR sequence to be output, and the read operation to be restarted (depending on the device code). The backspace character or CONTROL H causes the previous character entered to be ignored.

During write ASCII operations, the buffer is scanned to eliminate trailing blanks. Data is then sent to the VDU until the buffer is exhausted or until a carriage return is found in the data stream. A line feed is automatically appended to the detected carriage return; or if no carriage return is detected, an LF/CR sequence is sent to the terminal.

During image I/O, none of the above formatting actions occur. The amount of data requested is typed out or read in, without masking to 7-bit ASCII, eliminating trailing blanks, checking for backspace or CNTRL H characters, or detecting or appending carriage returns or line feeds. On image read, however, an ASCII CR is detected as an end of line sentinel.

STATUS	MEANING
X'C0'	Normal completion
X'82	Timeout or break
X'84'	Unrecoverable I/O error
X'A0'	Device unavailable

Depressing the break key while reading or writing causes X'82' status to be returned. This is fully compatible with the TTY keyboard printer driver; the ESC key has the same results.

When writing to the VDU in image mode, it is possible that the last character in the user buffer will be lost. Include an additional character in the buffer after the last valid character transmitted. Rubout, X'FF', is recommended. In format mode, a rubout character is automatically transmitted after the CR/LF sequence. This character does not appear on the screen.

5.14 8-LINE INTERRUPT MODULE

Interrupt simulation (SINT) is the only attribute supported by the Perkin-Elmer 8-line interrupt module. The module provides the processor with eight interrupt lines from external equipment and acknowledges interrupts on a priority basis. Any line can be selectively enabled or disabled. Several lines can be concurrently enabled. An interrupt does not transfer any data, nor is any status given.

5.15 DIGITAL MULTIPLEXOR

ASCII operations are not supported by the Perkin-Elmer digital multiplexor. During input, the second byte of the random address field contains the segment and point number to be read. Data is read from the point specified until the buffer specified by the starting and ending address is full.

During output, the second byte of the random address field contains the segment and point number to be written to. Data is written until the buffer specified by the starting and ending address is exhausted.

STATUS	MEANING
X'00'	Normal completion
X'A0'	Device unavailable
X'C0'	Illegal function; an ASCII operation was attempted
X'84'	Timeout
X'82'	Data transfer error; a nonexistent segment was selected

5.16 CONVERSION EQUIPMENT

The analog conversion equipment used with Perkin-Elmer 32-bit computers cannot be programmed in the device independent manner of other peripheral devices. The chassis, channel and card addresses, and data values are directly passed to the real time analog system controller as 16-bit words as they are obtained from the user.

During input, the random address field of the SVC 1 parameter block contains the starting address of a table containing analog-to-digital converter addresses (chassis address, channel address, and card address). The user buffer, which the start and end addresses of the parameter block determine, is loaded with the digitized data obtained from these analog-to-digital converters.

The table length containing the converter addresses, is equal to the length of the buffer. It is the user's responsibility to provide valid addresses. Since the analog input system mode of the controller is used for READ, if a nonexistent chassis is addressed, zero data is stored and no other indication is given.

During output, the user buffer is assumed to contain sequential pairs of alternating digital-to-analog converter addresses and the corresponding data to be converted; i.e., ADD1, DATA1, ADD2, DATA2,........ADDn, DATAn.

The address and data are directly passed to the real time analog system controller.

The control output mode of the controller is used for write operations. If a nonexistent chassis is addressed, the status is set to X'88', and the remainder of the I/O is aborted.

Each write sequence to any converter must consist of two halfwords. One halfword specifies the adapter to do the conversion; the other halfword contains the data to be converted. Thus, any attempt to do a write, with a buffer not a multiple of two halfwords in length, results in a memory fault.

For read and write operations, ASCII/binary and image/format requests are identical.

STATUS	MEANING
X'00'	Normal completion
X'C0'	Illegal function code
X'A0'	Device unavailable
X'84'	Unrecoverable error (hardware)
X'82'	Recoverable error; timeout (priority too low)
X'88'	Nonexistent chassis addressed on write
X'90'	Out of SYNC (priority too low)

5.17 ANALOG INPUT CONTROLLER

Variable record lengths are supported by the Perkin-Elmer analog input controller. ASCII operations are not supported. Command functions are ignored.

The random address field of the supervisor call 1 (SVC 1) parameter block contains the gain and address of the first channel to be sampled. The format is shown in Figure 5-1. Dividing the length of the user buffer (END-START+1) by two determines the number of channels to sample. The digitized data is sequentially stored in the user buffer, one halfword per channel.

```
 ---------------------------------------------------
 |   Gain    |                        |  Address   |
 ---------------------------------------------------
 Bits:
  0        3 4                      11  12       15
```

Figure 5-1 Random Field Format

STATUS	MEANING
X'C0'	Normal completion
X'C0'	Illegal function code
X'A0'	Device unavailable
X'82'	Recoverable error; device timeout
X'84'	Unrecoverable device error (device unavailable during transfer)
X'86'	Incorrect address alignment (START address odd, END address even)

The driver accepts only random calls, meaning that the first address is selected at random and that further addresses are sequential (in the same call). The start address must be on an even address boundary and the END address must be on an odd address boundary, since the analog input controller is a halfword device. This complies with Instrument Society of America (ISA) definition of sequential analog input.

5.18 ANALOG OUTPUT CONTROLLER

All command functions are ignored by the Perkin-Elmer analog output controller. One halfword of data is obtained from the user buffer in the format specified in Figure 5-2 and written to the device for conversion. This procedure is repeated until all halfwords in the user buffer are output. Dividing the length of the user buffer (END-START+1) by two computes the number of halfwords to be output.

```
 ---------------------------------------
|              |          Data          |
 ---------------------------------------
Bits:
 0            11 12                   15
```

Figure 5-2 Analog Output Data Format

STATUS	MEANING
X'C0'	Normal completion
X'C0'	Illegal function
X'86'	Incorrect address alignment (start address odd, end address even)

Binary image is treated identically to binary format; the image bit is ignored. The sequential/random bit is also ignored. The start address must be aligned on an even boundary; whereas, the end address must be on an odd boundary because the analog output controller is a halfword device.

5.19 DIGITAL I/O CONTROLLER

All command functions are ignored by the Perkin-Elmer digital I/O controller. The number of transfers is computed using the start and end address fields: (END-START+1)/2. Resetting the sequential/random bit in the function code field causes transfers to occur repeatedly, without interruption. This is a nonhandshaking transfer mode. In the handshaking transfer mode, the sequential/random bit is set, and each transfer occurs only after the internal strobe line is pulsed. A timeout rate for each transfer is set at a constant of four seconds.

During each binary read operation, the start address of the SVC 1 instruction points to a buffer that sequentially stores one halfword of data from the digital input card.

During binary write operations, the start address points to a buffer (K1) of halfwords consisting of image halfwords for output transfer. The random address field of the SVC 1 parameter block points to another buffer (K2) of halfwords designating masks that define what corresponding bit position in K1 is to be changed. The length of K2 must be the same as that of K1. A bit set in K2 indicates that the digital output is changed to the state defined by the corresponding bit position in K1. The following logical expression computes the halfwords transferred to the digital output card:

(K1.K2) + (K2.R)

Where:

. means logical AND;
+ means logical OR;
K2 means one's complement of K2;
R is the last known content of the output register.

STATUS	MEANING
X'00'	Normal completion
X'C0'	Illegal function
X'A0'	Device unavailable
X'82'	Recoverable error; timeout
X'84'	Unrecoverable error; device unavailable occurred during transfer
X'86'	Incorrect address alignment (start address odd, end address even, random address odd).

Binary image is treated identically to binary format; the drive ignores the formatted/image bit of the SVC 1 function code. The input and output sides are used in either the handshaking or nonhandshaking transfer modes. The start and random addresses must be aligned on the even boundaries; whereas, the end address must be aligned on an odd boundary because the digital I/O controller is a halfword device.

APPENDIX A
OS/32 SUPPORTED I/O DEVICES

TYPE	DEVICE	PRODUCT NUMBER	RD	WFT	TBS	EIN	WAT	FND	FLP
Card equipment	400 CPM card reader	M46-238/9	x			x			
	1000 CPM card reader	M46-236/7	x			x			
	High speed card reader/punch			x		x	x		
	High speed card reader/punch w/ separate print option		x	x		x	x		
Teletype (TTY) reader/ punch	Model 33 *		x	x		x	x		
	Model 35 *		x	x		x	x		
	Carousel 35 with paper tape reader 132-character line		x			x	x		
Teletype TTY keyboard printer	Model 33	M46-000/2/4/5	x	x			x		x
	Model 35	M46-001/3	x	x			x		x
	Perkin-Elmer Carousel * 15, 30, 35, 132-character line *	M46-010/1/5/6 M46-880	x	x			x		x
	Perkin-Elmer Carousel * 15, 30, 35, 80-character line *	M46-010/1/5/6	x	x			x		x
Paper tape equipment	Paper tape reader/ punch	M46-242/3	x	x		x	x		

Printers	Low speed line printer			x			x				
	Character printer	M46-221/2/3/4		x			x				
	Medium speed line printers	M46-300/1/2/3/4/5/6/7		x			x				
	High speed line printer			x			x				
	Thermal page printers	M46-066/8/and m46-080/2		x			x				
Tape Cassette	Intertape	M46-400	x	x		x	x		x		
Magnetic tape	800bpi		x	x		x	x		x		
	1600bpi		x	x		x	x		x		
	6250bpi		x	x		x	x		x		
	9-track, 75-ips, 800-bpi	M46-490/2	x	x		x	x		x		
	9-track, 45-ips, 800/1600-bpi	M46-494/6	x	x		x	x		x		
	9-track, 45-ips, 800-bpi	M46-501/2	x	x		x	x		x		
	9-track, 45-ips, 1600-bpi	M46-515/16	x	x		x	x		x		
Discs	2.5Mb removable disk		x	x	x	x	x	x			
	10Mb disk system (5Mb fixed, 5 Mb removable)		x	x	x	x	x	x			
	67Mb disk		x	x	x	x	x	x			
	256Mb disk		x	x	x	x	x	x			
	68.6Mb disk		x	x	x	x	x	x			
	MSM 300 disk system (300Mb drive and controller)		x	x	x	x	x	x			
	MSM 80 disk system (80Mb drive and controller	M46-600/2	x	x	x	x	x	x			

	MSM 80F disk system	M46-691/2	x	x	x	x	x	x		
	MSM 80F/HPT disk system	M46-693/4	x	x	x	x	x	x		
	Vanguard 1 cartridge disk system	M46-710/11	x	x	x	x	x	x		
Video display units (VDU)	Nonediting VDU $ ^		x	x			x			x
	Graphic display terminals $ ^		x	x			x			x
	Carousel 300 ^ $		x	x			x			x
	Carousel 300 with electronic format controls ^		x	x			x			x
	Model 1200 VDU		x	x			x			x
	Model 1100 VDU $ ^		x	x			x			x
	Model 550	M46-11-/111/ 112/113/114	x	x			x			x
	Model 1250	M46-121/2/3/4/ 15/6	x	x			x			x
	Model 1251	M46-47/8/9/and 50/51/52	x	x			x			x
Floppy disk	Floppy disk	M46	x	x		x	x	x		
8-line interrupt module	8-line interrupt module	M48-001	x							
Digital multiplexor	Digital multiplexor controller	M07-860	x	x		x	x	x		
Conversion equipment	Realtime analog system with user supplied extern-al clock	M48-603	x	x		x	x			
	Realtime analog system with user supplied external clock	M48-603	x	x		x	x			
Analog I/O controller	Mini I/O input	M48-212/3/4/5	x			x	x	x		
	Mini I/O output	M48-353/4/5		x		x	x	x		

Digital I/O controller	Mini I/O Module	M48-450	x	x		x	x	x			

LEGEND

* CLI - Current Loop Interface
^ CLCM - Current Loop Communications Multiplexor
$ RS232C

ATTRIBUTES

RD READ
WRT WRITE
TBAS TEST & SET
BIN BINARY
WAT WAIT
RND RANDOM
FLP FILE POSITION
INT INTERACTIVE
HLT HALT I/O

NOTE

Devices listed without a product number are no longer marketed by Perkin-Elmer.

APPENDIX B
POWERUP

INDEX