```
PPPPPPPPPPPP     LLL                 IIIIIIIII     RRRRRRRRRRRR     TTTTTTTTTTTTTT   LLL
PPPPPPPPPPPP     LLL                 IIIIIIIII     RRRRRRRRRRRR     TTTTTTTTTTTTTT   LLL
PPPPPPPPPPPP     LLL                 IIIIIIIII     RRRRRRRRRRRR     TTTTTTTTTTTTTT   LLL
PPP         PPP  LLL                    III        RRR         RRR        TTT        LLL
PPP         PPP  LLL                    III        RRR         RRR        TTT        LLL
PPP         PPP  LLL                    III        RRR         RRR        TTT        LLL
PPP         PPP  LLL                    III        RRR         RRR        TTT        LLL
PPP         PPP  LLL                    III        RRR         RRR        TTT        LLL
PPP         PPP  LLL                    III        RRR         RRR        TTT        LLL
PPPPPPPPPPPP     LLL                    III        RRRRRRRRRRRR           TTT        LLL
PPPPPPPPPPPP     LLL                    III        RRRRRRRRRRRR           TTT        LLL
PPPPPPPPPPPP     LLL                    III        RRRRRRRRRRRR           TTT        LLL
PPP              LLL                    III        RRR   RRR              TTT        LLL
PPP              LLL                    III        RRR   RRR              TTT        LLL
PPP              LLL                    III        RRR   RRR              TTT        LLL
PPP              LLL                    III        RRR      RRR           TTT        LLL
PPP              LLL                    III        RRR      RRR           TTT        LLL
PPP              LLL                    III        RRR      RRR           TTT        LLL
PPP              LLLLLLLLLLLLLLL     IIIIIIIII     RRR         RRR        TTT        LLLLLLLLLLLLLLL
PPP              LLLLLLLLLLLLLLL     IIIIIIIII     RRR         RRR        TTT        LLLLLLLLLLLLLLL
PPP              LLLLLLLLLLLLLLL     IIIIIIIII     RRR         RRR        TTT        LLLLLLLLLLLLLLL
```

FILEID**PLICHRSTR

```
PPPPPPPP    LL          IIIIII     CCCCCCCC  HH      HH  RRRRRRRR      SSSSSSSS  TTTTTTTTTT  RRRRRRRR
PPPPPPPP    LL          IIIIII     CCCCCCCC  HH      HH  RRRRRRRR      SSSSSSSS  TTTTTTTTTT  RRRRRRRR
PP      PP  LL            II      CC        HH      HH  RR      RR    SS            TT      RR      RR
PP      PP  LL            II      CC        HH      HH  RR      RR    SS            TT      RR      RR
PP      PP  LL            II      CC        HH      HH  RR      RR    SS            TT      RR      RR
PP      PP  LL            II      CC        HH      HH  RR      RR    SS            TT      RR      RR
PPPPPPPP    LL            II      CC        HHHHHHHHHH  RRRRRRRR      SSSSSS        TT      RRRRRRRR
PPPPPPPP    LL            II      CC        HHHHHHHHHH  RRRRRRRR      SSSSSS        TT      RRRRRRRR
PP          LL            II      CC        HH      HH  RR  RR              SS      TT      RR  RR
PP          LL            II      CC        HH      HH  RR  RR              SS      TT      RR  RR
PP          LL            II      CC        HH      HH  RR    RR            SS      TT      RR    RR      ....
PP          LL            II      CC        HH      HH  RR    RR            SS      TT      RR    RR      ....
PP          LLLLLLLLLL  IIIIII      CCCCCCCC  HH      HH  RR      RR  SSSSSSSS      TT      RR      RR      ....
PP          LLLLLLLLLL  IIIIII      CCCCCCCC  HH      HH  RR      RR  SSSSSSSS      TT      RR      RR      ....

LL            IIIIII      SSSSSSSS
LL            IIIIII      SSSSSSSS
LL              II      SS
LL              II      SS
LL              II      SS
LL              II      SS
LL              II        SSSSSS
LL              II        SSSSSS
LL              II              SS
LL              II              SS
LL              II              SS
LL              II              SS
LLLLLLLLLL  IIIIII    SSSSSSSS
LLLLLLLLLL  IIIIII    SSSSSSSS
```

PLI
VAX

Mac

-$2
-$2
TOT

234

The

MAC

Table of contents

```
0000     1         .title  pli$chrstr - character star returns routine
0000     2         .ident  /1-004/                                         ; Edit CGN1004
0000     3                                                                 ; Edit WHM1003
0000     4 ;
0000     5 ;*****************************************************************************
0000     6 ;*                                                                           *
0000     7 ;*  COPYRIGHT (c) 1978, 1980, 1982, 1984 BY                                  *
0000     8 ;*  DIGITAL EQUIPMENT CORPORATION, MAYNARD, MASSACHUSETTS.                   *
0000     9 ;*  ALL RIGHTS RESERVED.                                                     *
0000    10 ;*                                                                           *
0000    11 ;*  THIS SOFTWARE IS FURNISHED UNDER A LICENSE AND MAY BE USED AND COPIED    *
0000    12 ;*  ONLY IN  ACCORDANCE WITH  THE  TERMS  OF  SUCH  LICENSE  AND WITH THE   *
0000    13 ;*  INCLUSION OF THE ABOVE COPYRIGHT NOTICE. THIS SOFTWARE OR  ANY  OTHER    *
0000    14 ;*  COPIES THEREOF MAY NOT BE PROVIDED OR OTHERWISE MADE AVAILABLE TO ANY    *
0000    15 ;*  OTHER PERSON.  NO TITLE TO AND OWNERSHIP OF  THE  SOFTWARE IS  HEREBY    *
0000    16 ;*  TRANSFERRED.                                                             *
0000    17 ;*                                                                           *
0000    18 ;*  THE INFORMATION IN THIS SOFTWARE IS  SUBJECT TO CHANGE WITHOUT NOTICE    *
0000    19 ;*  AND  SHOULD  NOT  BE  CONSTRUED AS  A COMMITMENT BY DIGITAL EQUIPMENT   *
0000    20 ;*  CORPORATION.                                                             *
0000    21 ;*                                                                           *
0000    22 ;*  DIGITAL ASSUMES NO RESPONSIBILITY FOR THE USE  OR  RELIABILITY OF ITS    *
0000    23 ;*  SOFTWARE ON EQUIPMENT WHICH IS NOT SUPPLIED BY DIGITAL.                  *
0000    24 ;*                                                                           *
0000    25 ;*                                                                           *
0000    26 ;*****************************************************************************
0000    27 ;
0000    28 ;++
0000    29 ; facility:
0000    30 ;
0000    31 ;       VAX/VMS PL1 runtime library.
0000    32 ;
0000    33 ; abstract:
0000    34 ;
0000    35 ;       This module contains the routines necessary for functions to return
0000    36 ;       string values on the stack.
0000    37 ;
0000    38 ; author: r. heinen 18-apr-1979
0000    39 ;
0000    40 ; modified by r. heinen 5-Jan-1982 to fix bug when using CALLS and string
0000    41 ;                 target is more than 255 longwords away from current stack
0000    42 ;                 frame.
0000    43 ;
0000    44 ;       1-003   Bill Matthews   29-September-1982
0000    45 ;
0000    46 ;                 Invoke macros $defdat and rtshare instead of $defopr and share.
0000    47 ;
0000    48 ;       1-004   Chip Nylander   28-February-1983
0000    49 ;
0000    50 ;                 A   the PLI$CHARSTR_R6 routine to handle the case when
0000    51 ;                 a   rir} is returned through multiple levels of procedure
0000    52 ;                 invocation, in BEGIN-END blocks for example.
0000    53 ;
0000    54 ;--
0000    55
0000    56 ;
0000    57 ; external definitions
```

```
                                0000     58 ;
                                0000     59          $dscdef                         ; define descriptors
                                0000     60          $sfdef                          ; define stack offsets
                                0000     61
                                0000     62
                                0000     63          rtshare
                                0000     64 ;
                                0000     65 ; data table for calculation of register save area size
                                0000     66 ;
                                0000     67 table:
0C 08 08 04 0C 08 08 04 08 04 04 00     0000     68          .byte   0,4,4,8,4,8,8,12,4,8,8,12,8,12,12,16
                   10 0C 0C 08          000C
```

```
                                J 13
0010     70          .sbttl  return string function routines
0010     71 ;++
0010     72 ; pli$charstr_r6 - return string via unknown descriptor through multiple levels
0010     73 ;
0010     74 ; functional description:
0010     75 ;
0010     76 ; This routine is entered with a JMP instruction.
0010     77 ;
0010     78 ; The action of this routine is to return a string based on the function
0010     79 ; return argument addressed by r2.
0010     80 ;
0010     81 ; The routine either:
0010     82 ;
0010     83 ;       1. Returns the string via the specified descriptor if the address
0010     84 ;          field is filled in the descriptor.
0010     85 ;
0010     86 ;       2. Returns the string on the stack if the address field is null.
0010     87 ;
0010     88 ; The format of the returned storage is based on the descriptor type.
0010     89 ;
0010     90 ; At this time, two types are allowed:
0010     91 ;
0010     92 ;       1. dsc$k_class_s        - fixed character descriptor
0010     93 ;       2. dsc$k_class_vs       - varying character descriptor
0010     94 ;
0010     95 ; If the descriptor address field is valid the string is returned using the
0010     96 ; descriptor size field to specify the maximum size.  The descriptor is not
0010     97 ; written.
0010     98 ;
0010     99 ; If the address field of the descriptor is null then the string is returned
0010    100 ; on the stack and the descriptor size and address are filled to reflect the
0010    101 ; return value.
0010    102 ;
0010    103 ; In the case of a stack return, the caller regains control with SP addressing
0010    104 ; the data in the descriptor specified form.
0010    105 ;
0010    106 ; If r3 is zero, then all the work can be handled by a simple application
0010    107 ; of ots$charstar_r6, so we just jump there.  Otherwise, we unwind the
0010    108 ; appropriate number of stack frames as directed by r3, then jump to
0010    109 ; ots$charstar_r6 to finish up.
0010    110 ;
0010    111 ; If the string to be returned is in the local storage of a stack frame to be
0010    112 ; unwound, we allocate a LIB$GET_VM buffer to cache the string while
0010    113 ; things are being unwound, then copy it back to the local storage of the
0010    114 ; resulting procedure frame before jumping to ots$charstar_r6.
0010    115 ;
0010    116 ; Don't be tempted to get any fancier in the future.  All the fancy ideas
0010    117 ; were considered and discarded.  Trying to "bubble up" the string by
0010    118 ; successive invocation of ots$charstar_r6 from a shell routine doesn't work:
0010    119 ; cases like
0010    120 ;                       CALLS   #0,FOO
0010    121 ;                     .ENTRY    FOO,<R7,R8,R9>
0010    122 ;                       CALLS   #0,BAR
0010    123 ;                     .ENTRY    BAR <R2,R3,R4,R5,R6>
0010    124 ;                       JMP     PLI$CHARSTR_R6
0010    125 ;
0010    126 ; cause registers R?-R6 to be permanently obliterated by the second trip
```

```
                     0010   127 ; through ots$charstar_r6.  Trying to manually restore all the proper registers
                     0010   128 ; from the proper places as represented by the intervening procedure frames
                     0010   129 ; was hopeless.  Attempting to slide the intervening procedure frames
                     0010   130 ; down to make room for the string in the calling procedure creates small
                     0010   131 ; windows of stack-inconsistency that interfere with AST handling and
                     0010   132 ; VMS checkpointing.
                     0010   133 ;
                     0010   134 ; inputs:
                     0010   135 ;
                     0010   136 ;       r0 = size of the string to return
                     0010   137 ;       r1 = address of the string to return
                     0010   138 ;       r2 = address of the return descriptor
                     0010   139 ;       r3 = number of procedure frames to unwind
                     0010   140 ;
                     0010   141 ; outputs:
                     0010   142 ;
                     0010   143 ;       none.
                     0010   144 ;
                     0010   145 ; exceptions:
                     0010   146 ;
                     0010   147 ;       A lib$get_vm or lib$free_vm failure generates an appropriate
                     0010   148 ;       lib$signal.  See also ots$charstar_r6
                     0010   149 ;
                     0010   150 ;--
                     0010   151 pli$charstr_r6::
           53   D5  0010   152         tstl    r3                      ; generated code should never do this,
           03   12  0012   153         bneq    10$                     ; but check just to make sure
         00A2   31  0014   154         brw     ots$charstar_r6         ; no work to do
      56   5D   D0  0017   155 10$:    movl    fp,r6                   ; initialize for frame search
           53   D7  001A   156 20$:    decl    r3                      ; find last frame to be unwound
           06   15  001C   157         bleq    30$                     ; branch if done
      56   0C A6   D0  001E   158         movl    sf$l_save_fp(r6),r6     ; pickup next fp
           F6   11  0022   159         brb     20$                     ; loop
         53   50   7D  0024   160 30$:    movq    r0,r3                   ; save string size and address
         55   10   D0  0027   161         movl    #16,r5                  ; initialize buffer size
         5E   54   D1  002A   162         cmpl    r4,sp                   ; is string to return on stack?
              08   19  002D   163         blss    40$                     ; if lss then no
         56   54   D1  002F   164         cmpl    r4,r6                   ; is it in a frame to unwind?
              03   14  0032   165         bgtr    40$                     ; if gtr then no
         55   53   C0  0034   166         addl    r3,r5                   ; have to cache string, add size
              55   DD  0037   167 40$:    pushl   r5                      ; push number of bytes to allocate
              7E   D4  0039   168         clrl    -(sp)                   ; address to return address
              6E   9F  003B   169         pushab  (sp)                    ; setup argument list
           08 AE   9F  003D   170         pushab  8(sp)                   ;
00000000'GF   02   FB  0040   171         calls   #2,g^lib$get_vm         ; allocate the memory
           0A 50   E8  0047   172         blbs    r0,50$                  ; if low not set ther error
              50   DD  004A   173         pushl   r0                      ; signal error condition
00000000'GF   01   FB  004C   174         calls   #1,g^lib$signal
                   04  0053   175         ret                             ; really can't finish after this
              50 8ED0  0054   176 50$:    popl    r0                      ; get address of allocated space
      08 A6   50   D0  0057   177         movl    r0,sf$l_save_ap(r6)     ; save in ap of last frame to unwind
              80 8ED0  005B   178         popl    (r0)+                   ; save buffer size in buffer
         80   53   7D  005E   179         movq    r3,(r0)+                ; save string size and length in buffer
         80   52   D0  0061   180         movl    r2,(r0)+                ; save desc address in buffer
         55   10   D1  0064   181         cmpl    #16,r5                  ; do we need to cache the string?
              04   13  0067   182         beql    60$                     ; if eql then no
   60   64   53   28  0069   183         movc3   r3,(r4),(r0)            ; copy string to buffer
```

```
                50   0000007E'EF   9E   006D   184 60$:    movab   70$,r0                  ; want to continue here
                      51   0C A6   D0   0074   185          movl    sf$l_save_fp(r6),r1     ; in this procedure context
                     00000000'GF   17   0078   186          jmp     g^pli$nonloc_goto       ; unwind what we don't want
                         6C   10   D1   007E   187 70$:    cmpl    #16,(ap)                ; do we need to restore the string?
                              0E   13   0081   188          beql    80$                     ; if eql then no
                        5E   04 AC   C2   0083   189          subl    4(ap),sp                ; make room on stack for string
             6E   10 AC   04 AC   28   0087   190          movc3   4(ap),16(ap),(sp)       ; restore string
                   08 AC   5E   D0   008D   191          movl    sp,8(ap)                ; update string address
                           8C   DD   0091   192 80$:    pushl   (ap)+                   ; push number of bytes to deallocate
                        FC AC   9F   0093   193          pushab  -4(ap)                  ; address to deallocate
                      53   8C   7D   0096   194          movq    (ap)+,r3                ; restore string size and address
                      52   8C   D0   0099   195          movl    (ap)+,r2                ; restore descriptor address
                           6E   9F   009C   196          pushab  (sp)                    ; setup argument list
                        08 AE   9F   009E   197          pushab  8(sp)                   ;
            00000000'GF   02   FB   00A1   198          calls   #2,g^lib$free_vm        ; deallocate the memory
                      09   50   E8   00A8   199          blbs    r0,90$                  ; if low not set then error
                           50   DD   00AB   200          pushl   r0                      ; signal error condition
            00000000'GF   01   FB   00AD   201          calls   #1,g^lib$signal
                      50   53   7D   00B4   202 90$:    movq    r3,r0                   ; setup for ots$charstar_r6
                           00   11   00B7   203          brb     ots$charstar_r6         ; and go finish up
                                     00B9   204
```

```
00B9    206 ;++
00B9    207 ; ots$charstar_r6 - return string via unknown descriptor
00B9    208 ;
00B9    209 ; functional description:
00B9    210 ;
00B9    211 ; This routine is entered with a JMP instruction.
00B9    212 ;
00B9    213 ; The action of this routine is to return a string based on the function
00B9    214 ; return argument addressed by r2.
00B9    215 ;
00B9    216 ; The routine either:
00B9    217 ;
00B9    218 ;       1. Returns the string via the specified descriptor if the address
00B9    219 ;          field is filled in the descriptor.
00B9    220 ;
00B9    221 ;       2. Returns the string on the stack if the address field is null.
00B9    222 ;
00B9    223 ; The format of the returned storage is based on the descriptor type.
00B9    224 ;
00B9    225 ; At this time, two types are allowed:
00B9    226 ;
00B9    227 ;       1. dsc$k_class_s        - fixed character descriptor
00B9    228 ;       2. dsc$k_class_vs       - varying character descriptor
00B9    229 ;
00B9    230 ; If the descriptor address field is valid the string is returned using the
00B9    231 ; descriptor size field to specify the mamimum size.  The descriptor is not
00B9    232 ; written.
00B9    233 ;
00B9    234 ; If the address field of the descriptor is null then the string is returned
00B9    235 ; on the stack and the descriptor size and address are filled to reflect the
00B9    236 ; return value.
00B9    237 ;
00B9    238 ; In the case of a stack return, the caller regains control with SP addressing
00B9    239 ; the data in the descriptor specified form.
00B9    240 ;
00B9    241 ; In order to shuffle the stack, a calculation is done which results in the
00B9    242 ; storage address of the string taking into account the CALLS notion of the
00B9    243 ; incoming call and the alignment of the stack prior to the call.
00B9    244 ; With this number, a new frame is created far enough below the other to allow
00B9    245 ; for the string.  The current frame and the subroutine frame are then merged
00B9    246 ; allowing for the saved registers in the previous frame. The current frame is
00B9    247 ; then changed to a CALLS frame with enough arguments and allignment such that
00B9    248 ; after the RET, SP is correctly addressing the return.
00B9    249 ; The previous frame is then removed from the stack FP list. With the space
00B9    250 ; reserved, the string  is copied to the target place being careful not to
00B9    251 ; write over the potential string data space.
00B9    252 ;
00B9    253 ; One optimization is possible.  If the incomming call is a CALLS then a
00B9    254 ; check is made to see if the argument list passed is large enough to
00B9    255 ; hold the target string.  If it is then no aditional frame need be created
00B9    256 ; and the string can be stored over the now worthless argument list.  SP is
00B9    257 ; then adjusted by modifying the argument count and frame alignment.
00B9    258 ;
00B9    259 ; inputs:
00B9    260 ;
00B9    261 ;       r0 = size of the string to return
00B9    262 ;       r1 = address of the string to return
```

```
                                00B9   263 ;       r2 = address of the return descriptor
                                00B9   264 ;
                                00B9   265 ; outputs:
                                00B9   266 ;
                                00B9   267 ;       none.
                                00B9   268 ;
                                00B9   269 ; exceptions:
                                00B9   270 ;
                                00B9   271 ; Any number of exceptions are possible during this operation. The stack
                                00B9   272 ; movement is done is a way that preserves the integrity of the stack but
                                00B9   273 ; does not provide a method for fielding exceptions and cleaning up.  No
                                00B9   274 ; data is stored if an exception occurs. If an exception handler fields the
                                00B9   275 ; exception and then unwinds to the caller the exception handler is in error.
                                00B9   276 ;
                                00B9   277 ;--
                                00B9   278 ots$charstar_r6::
              55    04 A2   D0  00B9   279         movl    dsc$a_pointer(r2),r5    ; fetch target arddress
                      04   13  00BD   280         beql    stack_return            ; if eql then return on stack
                    0106   30  00BF   281         bsbw    return_characters       ; return strings via descriptor
                      04   00C2   282         ret                             ; return to caller
                                00C3   283 ;
                                00C3   284 ; return varying string on stack - calcu ate target address
                                00C3   285 ;
                                00C3   286 stack_return:
                                00C3   287 ;
                                00C3   288 ; calculate the sp value prior to call taking into account the call
                                00C3   289 ; instruction type and previous stack alignment
                                00C3   290 ;
            54    06 AD   B0  00C3   291         movw    sf$w_save_mask(fp),r4   ; fetch frame's save mask
       55   54    02   0E   EF  00C7   292         extzv   #sf$v_stackoffs,#sf$s_stackoffs,r4,r5; get alignment
            55    14 AD45   9E  00CC   293         movab   20(fp)[r5],r5           ; address past fixed part of frame
                     56   D4  00D1   294         clrl    r6                      ; accumulate reg save mask size
       53   54    04   00   EF  00D3   295         extzv   #0,#4,r4,r3             ; calc size of register save mask
            53  FF23 CF43   9A  00D8   296         movzbl  w^table[r3],r3          ; get size in bytes for this part
                     56   53  C0  00DE   297         addl    r3,r6                   ; accumulate result
       53   54    04   04   EF  00E1   298         extzv   #4,#4,r4,r3             ; get byte size for next section
            53  FF15 CF43   9A  00E6   299         movzbl  w^table[r3],r3          ;
                     56   53  C0  00EC   300         addl    r3,r6                   ; accumulate total
       53   54    04   08   EF  00EF   301         extzv   #8,#4,r4,r3             ; get byte size for next section
            53  FF07 CF43   9A  00F4   302         movzbl  w^table[r3],r3          ;
                     56   53  C0  00FA   303         addl    r3,r6                   ; accumulate total
                     55   56  C0  00FD   304         addl    r6,r5                   ; calc address based on size
                                0100   305 ;
                                0100   306 ; accumulate argument list space if CALLS used
                                0100   307 ;
                                0100   308 ; check for possible optimization noted above
                                0100   309 ;
                          52    DD  0100   310         pushl   r2                      ; save address of the input descriptor
                     52   55   D0  0102   311         movl    r5,r2                   ; copy possible address of arglist
                     55   50   C2  0105   312         subl    r0,r5                   ; calc string target address
                     55   02   C2  0108   313         subl    #2,r5                   ; allocate size word space
                34   54   0D   E1  010B   314         bbc     #sf$v_calls,r4,move_frame; br if not calls
                53   82   02   78  010F   315         ashl    #2,(r2)+,r3             ; fetch size of calls arguments in bytes
                     55   53   C0  0113   316         addl    r3,r5                   ; calc original caller's sp
                     55   04   C0  0116   317         addl    #4,r5                   ; include argcount longword
                     02   0E   EF  0119   318         extzv   #sf$v_stackoffs,#sf$s_stackoffs,-
                  53    06 AD       011C   319                 sf$w_save_mask(fp),r3   ;get this frames alignment
```

```
                52   53    C2  011F    320          subl    r3,r2                   ;align future ap start
             53 55   52    C3  0122    321          subl3   r2,r5,r3                ; is target within arglist?
                     1B    19  0126    322          blss    move_frame              ; if lss then no - use elaborate return
                           0128    323 ;
                           0128    324 ; string can be returned inside the calls argument list
                           0128    325 ;
          54    53   02    00  EF  0128    326          extzv   #0,#2,r3,r4             ; get byte offset value
                02   0E    54  F0  012D    327          insv    r4,#sf$v_stackoffs,#sf$s_stackoffs,-; set new alignment
                        06  AD      0131    328                  sf$w_save_mask(fp)      ;
                52   54    C0  0133    329          addl    r4,r2                   ; address future ap data
              72    53   FE  8F  78  0136    330          ashl    #-2,r3,-(r2)            ; insert new argument list size
                5C   52    D0  013B    331          movl    r2,ap                   ; reset ap address
                     52 8ED0   013E    332          popl    r2                      ; restore descriptor address
                     5B    11  0141    333          brb     unknown_target          ;
                           0143    334 ;
                           0143    335 ; setup area to return string to on stack
                           0143    336 ;
                           0143    337 ; allocate enough space for the string on the stack
                           0143    338 ;
                           0143    339 move_frame:
                     52 8ED0   0143    340          popl    r2                      ; restore descriptor size
                5E   55    D1  0146    341          cmpl    r5,sp                   ; stack deep enough now?
                     03    1E  014A    342          bgequ   15$                     ; if target above sp then yes
                5E   55    D0  014B    343          movl    r5,sp                   ; else start at targe. address
                           014E    344 ;
                           014E    345 ; create a stack frame below target area
                           014E    346 ;
                5E   03    CA  014E    347 15$:     bicl    #3,sp                   ; align stack
                53   7E    7E  0151    348          movaq   -(sp),r3                ; setup future argument list storage
                           0154    349                                             ; this storage will be used to base
                           0154    350                                             ; an argument list that will re-align
                           0154    351                                             ; the stack after final RET
                5E   56    C2  0154    352          subl    r6,sp                   ; save room for registers
          5D'AF   04  A3   FA  0157    353          callg   4(r3),b^20$             ; generate callg stack
                           015C    354                                             ; pass ap as future argument list address
                     04    015C    355          ret                             ; never used
                           015D    356 ;
                           015D    357 ; routine to copy string to stack
                           015D    358 ;
                           015D    359 ; inputs:
                           015D    360 ;
                           015D    361 ;       r0,r1 describe the string to return
                           015D    362 ;       r2 = address of the descriptor
                           015D    363 ;       r5 = address of the target
                           015D    364 ;       r6 = size of register save area
                           015D    365 ;       ap = address of future sp correction argument list
                           015D    366 ;
                           015D    367 ; after entry to this routine the current stack frame has the prevoius
                           015D    368 ; frame's registers saved in the same place as if the previous frame's
                           015D    369 ; register save mask was used for this entry.  By copying the register
                           015D    370 ; save mask, psw, and saved fp,ap,pc the frame will be identical.
                           015D    371 ;
                     0000  015D    372 20$:     .word   0                       ; registers saved above
                           015F    373 ;
                           015F    374 ; copy data from previous stack
                           015F    375 ;
             53   0C  AD   D0  015F    376          movl    sf$l_save_fp(fp),r3     ; address previous stack frame
```

```
                04 AD    04 A3    B0  0163    377          movw    sf$w_save_psw(r3),sf$w_save_psw(fp); insert psw
                06 AD    06 A3    B0  0168    378          movw    sf$w_save_mask(r3),sf$w_save_mask(fp); insert new save mask
                10 AD    10 A3    D0  016D    379          movl    sf$l_save_pc(r3),sf$l_save_pc(fp); copy pc from previous
                08 AD    08 A3    7D  0172    380          movq    sf$l_save_ap(r3),sf$l_save_ap(fp); copy original ap and ap
                                           0177    381                                          ; remove previous from stack list
                            27    BB  0177    382          pushr   #^m<r0,r1,r2,r5>             ; save registers
          14 AD    14 A3    56    28  0179    383          movc3   r6,20(r3),20(fp)             ; copy register save area
                            27    BA  017F    384          popr    #^m<r0,r1,r2,r5>             ; restore registers
             53    55       5C    C3  0181    385          subl3   ap,r5,r3                     ; calc extra arglist space for sp align
          54 55    02       00    EF  0185    386          extzv   #0,#2,r5,r4                  ; get byte alignment value
             02    0E       54    F0  018A    387          insv    r4,#sf$v_stackoffs,#sf$s_stackoffs,-; set new alignement
                      06 AD          018E    388                  sf$w_save_mask(fp)          ;
                   5C    54    C0  0190    389          addl    r4,ap                        ; point to new arglist start
             7C    53    FE    8F  0193    390          ashl    #-2,r3,-(ap)                 ; divide by 4 to get longwords
             06 AD       2000  8F  0198    391          bisw    #1@sf$v_calls,sf$w_save_mask(fp); force calls
                                           019E    392
                                           019E    393 ;
                                           019E    394 ; fi.l in the descriptor
                                           019E    395 ;
                                           019E    396 urknown_target:
                   62    50    B0  019E    397          movw    r0,dsc$w_length(r2)          ; return size
                04 A2    55    D0  01A1    398          movl    r5,dsc$a_pointer(r2)         ; specify address of target
                            21    10  01A5    399          bsbb    return_characters            ; move string
          50    6C    000000FF  8F  CB  01A7    400          bicl3   #^xff,(ap),r0                ; get arg count above 255
                            16    13  01AF    401          beql    25$                          ; if eql then leq 255
                   51    6C40    DE  01B1    402          moval   (ap)[r0],r1                  ; cal new ap address
                                           01B5    403                                          ; within 255 range
                   61    6C      9A  01B5    404          movzbl  (ap),(r1)                    ; insert new reduced arg count
                                           01B8    405                                          ; at future new ap site
                52 5C    5D      C3  01B8    406          subl3   fp,ap,r2                     ; cal size of current frame
                56 51    52      C3  01BC    407          subl3   r2,r1,r6                     ; cal address for new frame
                66 6D    52      28  01C0    408          movc3   r2,(fp),(r6)                 ; copy frame to new location
                   5D    56      D0  01C4    409          movl    r6,fp                        ; run on new frame
                         04      01C7    410 25$:     ret                                      ; return to caller
                                           01C8    411 ;
                                           01C8    412 ; case on target format
                                           01C8    413 ;
                                           01C8    414 return_characters:
                                           01C8    415          case    type=b,dsc$b_class(r2),<-
                                           01C8    416                  50$,-                ; invalid return
                                           01C8    417                  60$,-                ; dsc$k_class_s - fixed string
                                           01C8    418                  50$,-                ; invalid return
                                           01C8    419                  50$,-                ; invalid return
                                           01C8    420                  50$,-                ; invalid return
                                           01C8    421                  50$,-                ; invalid return
                                           01C8    422                  50$,-                ; invalid return
                                           01C8    423                  50$,-                ; invalid return
                                           01C8    424                  50$,-                ; invalid return
                                           01C8    425                  50$,-                ; invalid return
                                           01C8    426                  50$,-                ; invalid return
                                           01C8    427                  70$,-                ; dsc$k_class_vs - varying string
                                           01C8    428                  >
                         05      01E5    429 50$:     rsb                                      ; invalid type code
                                           01E6    430 ;
                                           01E6    431 ; fixed return format
                                           01E6    432 ;
                                           01E6    433 60$:
```

```
                65   61   50   28  01E6   434          movc3   r0,(r1),(r5)            ; return string
                                  05  01EA   435          rsb
                                      01EB   436 ;
                                      01EB   437 ; character varying return code
                                      01EB   438 ;
                                      01EB   439 70$:
                     62   50   B1  01EB   440          cmpw    r0,dsc$w_length(r2)     ; calc size to return
                          03   15  01EE   441          bleq    75$                     ; if leq then original in range
                     50   62   B0  01F0   442          movw    dsc$w_length(r2),r0     ; set desc as max size
                     62   50   B0  01F3   443 75$:     movw    r0,dsc$w_length(r2)     ; set size in desc
                          21   BB  01F6   444          pushr   #^m<r0,r5>              ; save size and target
              02 A5   61   50   28  01F8   445          movc3   r0,(r1),2(r5)           ; copy string
                                      01FD   446 ;
                                      01FD   447 ; for the elaborate case...
                                      01FD   448 ; the size must be returned after move to prevent possible overlap problem
                                      01FD   449 ;
                     9E   8E   F7  01FD   450          cvtlw   (sp)+,@(sp)+            ; return size at front of string
                                  05  0200   451          rsb
                                      0201   452          .end
```

```
DSC$A_POINTER                   = 00000004
DSC$B_CLASS                     = 00000003
DSC$W_LENGTH                    = 00000000
LIB$FREE_VM                       ********    X   02
LIB$GET_VM                        ********    X   02
LIB$SIGNAL                        ********    X   02
MOVE_FRAME                        00000143 R      02
OTS$CHARSTAR_R6                   000000B9 RG     02
PLI$CHARSTR_R6                    00000010 RG     02
PLI$NONLOC_GOTO                   ********      X 02
RETURN_CHARACTERS                 000001C8 R      02
SF$L_SAVE_AP                    = 00000008
SF$L_SAVE_FP                    = 0000000C
SF$L_SAVE_PC                    = 00000010
SF$S_STACKOFFS                  = 00000002
SF$V_CALLS                      = 0000000D
SF$V_STACKOFFS                  = 0000000E
SF$W_SAVE_MASK                  = 00000006
SF$W_SAVE_PSW                   = 00000004
STACK_RETURN                      000000C3 R      02
TABLE                             00000000 R      02
UNKNOWN_TARGET                    0000019E R      02
```

Psect synopsis

PSECT name	Allocation	PSECT No.	Attributes
. ABS .	00000000 (0.)	00 (0.)	NOPIC USR CON ABS LCL NOSHR NOEXE NORD NOWRT NOVEC BYTE
ABS	00000000 (0.)	01 (1.)	NOPIC USR CON ABS LCL NOSHR EXE RD WRT NOVEC BYTE
_PLI$CODE	00000201 (513.)	02 (2.)	PIC USR CON REL LCL SHR EXE RD NOWRT NOVEC LONG

Performance indicators

Phase	Page faults	CPU Time	Elapsed Time
Initialization	13	00:00:00.07	00:00:01.33
Command processing	82	00:00:00.55	00:00:04.01
Pass 1	116	00:00:02.71	00:00:09.56
Symbol table sort	0	00:00:00.21	00:00:00.84
Pass 2	83	00:00:01.03	00:00:04.72
Symbol table output	3	00:00:00.03	00:00:00.03
Psect synopsis output	2	00:00:00.02	00:00:00.02
Cross-reference output	0	00:00:00.00	00:00:00.00
Assembler run totals	299	00:00:04.62	00:00:20.51

The working set limit was 900 pages.
15616 bytes (31 pages) of virtual memory were used to buffer the intermediate code.
There were 10 pages of symbol table space allocated to hold 171 non-local and 18 local symbols.
452 source lines were read in Pass 1, producing 11 object records in Pass 2.
11 pages of virtual memory were used to define 10 macros.

```
                                    +----------------------------+
                                    ! Macro library statistics !
                                    +----------------------------+

Macro library name                                  Macros defined
------------------                                  --------------
_$255$DUA28:[PLIRTL.OBJ]PLIRTMAC.MLB;1                         2
_$255$DUA28:[SYSLIB]STARLET.MLB;2                              5
TOTALS (all libraries)                                         7

234 GETS were required to define 7 macros.

There were no errors, warnings or information messages.

MACRO/ENABLE=SUPPRESSION/DISABLE=TRACEBACK/LIS=LIS$:PLICHRSTR/OBJ=OBJ$:PLICHRSTR MSRC$:PLICHRSTR/UPDATE=(ENH$:PLICHRSTR)+LIB$:PLIRTM
```

PLICASERR
LIS

PLICONDIT
LIS

DATDEF
SDL

FILEDEF
IN

PLIRTDEF
MAR

PLICHRSTR
LIS

PLICHAR
LIS

PLICLOSE
LIS

PLIRTL

PLIFILDSP
IN

PLIBYTSIZ
LIS

PLIRTL
MAP

ENVCODES
IN

PLIBIT
LIS