DataGeneral
Data General Corporation, Westboro, Massachusetts 01580

Customer Documentation

AOS/VS, AOS/VS II, and AOS/RT32 System Call Dictionary, ?A Through ?Q

093-000542-02

AOS/VS, AOS/VS II, and AOS/RT32 System Call Dictionary, ?A Through ?Q

093-000542-02

For the latest enhancements, cautions, documentation changes, and other information on this product, please see the Release Notice (085-series) supplied with the software.

Ordering No. 093-000542

Printed in the United States of America
Rev. 02, December, 1991

Notice

AOS/VS, AOS/VS II, and AOS/RT32 System Call Dictionary, ?A Through ?Q
093-000542-02

Revision History:
Original Release – October 1988
First Revision – February 1990
Second Revision – December 1991
Addendum 086-000195 – June 1992

Effective with:

AOS/VS, Rel. 7.70
AOS/VS II, Rel. 2.20
AOS/RT32, Rel. 5.01

A vertical bar in the outer margin of a page indicates substantive change from the previous revision of this manual.

DataGeneral

Addendum

Addendum to AOS/VS, AOS/VS II, and AOS/RT32 System Call Dictionary, ?A Through ?Q

086-000195-00

This addendum updates your manual 093-000542-02. Please see "Updating Your Manual." If you are running AOS/VS Revision 7.69, do not insert this addendum, which becomes effective with AOS/VS Revision 7.70.

Printed in the United States of America
Revision 00, June 1992
Licensed Material – Property of Data General Corporation
Ordering No. 086-000195

Notice

Addendum to
AOS/VS, AOS/VS II, and AOS/RT32 System Call Dictionary, ?A Through ?Q
086-000195-00

In the margins of replacement pages, a vertical bar indicates substantive technical change from 093-000542-02.

The addendum number appears on all pages in this addendum.

Updating Your Manual

This addendum (086–000195–00) to AOS/VS, AOS/VS II, and AOS/RT32 System Call Dictionary, ?A Through ?Q introduces new information effective with AOS/VS II Release 2.20, and AOS/VS Release 7.70. It also includes minor corrections.

To update your copy of 093–000542–02, please remove manual pages and insert addendum pages as follows:

Remove	Insert
Title/Notice	Title/Notice
iii through vi	iii through vi
old Contents	new Contents
2–11 through 2–20	2–11 through 2–20
2–27/2–28	2–27/2–28
2–57/2–58	2–57/58 through 2–58.26
2–69/2–70	2–69/2–70
2–85/2–86	2–85/2–86
2–89/2–90	2–89/2–90
2–103/2–104	2–103/2–104
2–113/2–114	2–113/2–114
2–121/2–122	2–121/2–122
2–159 through 2–164	2–159 through 2–164
2–171 through 2–174	2–171 through 2–174
2–203 through 2–216	2–203 through 2–216
2–259/2–260	2–259/2–260
2–269 through 2–272	2–269 through 2–272
2–287/2–288	2–287/2–288
2–295 through 2–304	2–295 through 2–304.9
2–307/2–308	2–307/2–308
2–337/2–338	2–337/2–338
2–405 through 2–410	2–405 through 2–410
2–415 through 2–418	2–415 through 2–418
2–473/2–474	2–473/2–474
2–525 through 2–530	2–525 through 2–530
2–535 through 2–548	2–535 through 2–548
2–555/2–556	2–555/2–556

Remove	Insert
old Index	new Index
old Document Set	new Document Set

Where new material requires additional pages, the pages have a decimal and number suffix; for example 5-21.1, 5-22.2.

Insert this updating sheet immediately behind the new Title/Notice page.

Preface

The System Call Dictionary spans two manuals — one for system calls ?A through ?Q, and the other for system calls ?R through ?Z. Much information appears twice in these two manuals for your ease of use. For example, the table of contents and indexes are identical. Chapter 2 in both books has the same title, but their contents differ. Chapter 2 in manual 093–000543 is a continuation of Chapter 2 in manual 093–000542, and is paginated accordingly. Appendixes A and B follow Chapter 2 in the second manual.

This manual is intended for use by experienced assembly language programmers. Experienced high–level language programmers can also use this manual to create programs that make direct calls to the operating systems.

Organization

This manual is organized as follows:

Chapter 2 begins with a summary table of all AOS/VS and AOS/RT32 system calls, followed by detailed descriptions of all the system calls whose names begin with ?A through ?Q.

Appendix A contains 12 program sets that illustrate AOS/VS and AOS/RT32 system calls. We have written 11 of the program sets in assembly language and the twelfth in FORTRAN 77. The Appendix is located at the end of the complementary manual *AOS/VS, AOS/VS II, and AOS/RT32 System Call Dictionary, ?R Through ?Z* (093–000543).

Appendix B describes the format of the *system log (SYSLOG) file*, into which both AOS/VS and AOS/VS II and privileged processes can write records that log the occurrence of certain events. The Appendix is located at the end of the complementary manual *AOS/VS, AOS/VS II, and AOS/RT32 System Call Dictionary, ?R Through ?Z* (093–000543).

Related Documentation

As mentioned earlier, the complement of this manual is *AOS/VS, AOS/VS II, and AOS/RT32 System Call Dictionary, ?R Through ?Z* (093–000543). The following documents are ancillary to both manuals.

- *AOS/VS System Concepts* (093–000335)
- *Introduction to AOS/RT32* (069–400061)
- *AOS/VS and AOS/VS II Error and Status Messages* (093–000540)
- *AOS/VS and AOS/VS II Glossary* (069–000231)

AOS/VS System Concepts and *Introduction to AOS/RT32*, listed at the beginning of this section, contain a general description of operating system calls and how to use them. This manual and its companion system call dictionary manual contain detailed descriptions of each AOS/VS and AOS/RT32 call. For your convenience, the system call descriptions in the two dictionaries are in alphabetical order.

The Documentation Set, after the index in each manual, contains a complete annotated list of AOS/VS and AOS/VS II manuals.

If you are not experienced with assembly language, we suggest that you read the following manuals before you read this book:

- *Fundamentals of Small Computer Programming* (093–000090), which provides a general introduction to Data General computers.
- *AOS/VS Macroassembler (MASM) Reference Manual* (093–000242), which gives detailed information about the syntax of AOS/VS assembly language and about the Macroassembler utility.
- *ECLIPSE® MV/Family (32–Bit) Systems Principles of Operation* (014–001371), which explains the processor–independent concepts and functions of ECLIPSE® MV/Family systems to assembly language programmers.
- *ECLIPSE® MV/Family (32–Bit) Systems Instruction Dictionary* (014–001372), which explains each instruction in the ECLIPSE MV/Family instruction set to assembly language programmers. Processor–dependent information, available in machine–specific supplements, complements this and the previous manual. An example of such information is found in the manual *ECLIPSE® MV/20000™ Series Systems Principles of Operation Supplement* (014–001169).
- *ECLIPSE® MV/Family Instruction Reference Booklet* (014–000702), which provides a brief summary of the instruction set and register information. The reference booklet lists each instruction by assembler–recognizable mnemonic with a shorthand description of its function.
- *FORTRAN 77 Environment Manual (AOS/VS)* (093–000288).

Update and Release Notices

Certain features of the operating systems may change from revision to revision. Therefore, please refer to the current Release Notice for the most up–to–date information about functional changes and enhancements. The Release Notice is usually in the utilities directory (:UTIL) on your system. The filename of the AOS/VS Model 3900 Update Notice is 078_000105_**; that of the AOS/VS II Release Notice is 085_000930_**. Suffixes (**) change with each revision. Your system manager should be able to tell you the exact pathname of the Release Notice.

The AOS/VS and AOS/VS II Release and Update Notices contain the latest details about all the system software, including enhancements and changes, notes and warnings. Notices are supplied both as printed listings and as disk files that you can print. The manuals and the Notices comprise the documentation for the system calls for AOS/VS Revision 7.69, and for AOS/VS II Revision 2.10. There are no documentation–changes files for this manual.

You should read the Update and Release Notices. If you want to know the features of a release, or have problems with a release, read the notice for solutions. The notices assume that you know the operating system well — so parts of the notices may be difficult to understand until you *do* know the system.

The Newsletter

Finally, you will find the *AOS/VS Monthly Newsletter* a useful source of information on the latest enhancements to both AOS/VS and AOS/VS II.

Reader Please Note

Throughout this manual we use the following format conventions:

COMMAND required *[optional]* ...

Where	Means
COMMAND	You must enter the command (or its accepted abbreviation) as shown.
required	You must enter some argument (such as a filename). Sometimes, we use $\left\{\begin{matrix}\text{required}_1\\ \text{required}_2\end{matrix}\right\}$ which means you must enter one of the arguments. Do not type the braces; they only set off the choices.
[optional]	You have the option of entering this argument. Do not type the brackets; they only identify the argument as an option.
...	You may repeat the preceding entry.

Standard Symbols

Additionally, we use the following symbols:

Symbol	Means
↵	Press the New Line, Carriage Return (CR), or Enter key on your terminal keyboard.
)	The CLI prompt.
< >	Angle brackets indicate the paraphrase of an argument or statement. (You supply the actual argument or statement.)
*	One asterisk indicates multiplication. For example, 2*3 means 2 multiplied by 3.
**	Two asterisks indicate exponentiation. For example, 2**3 means 2 raised to the third power.
OS	The operating system in the accumulator I/O, figure, and table categories.

Unless the text supplies a specific radix (as it often does), all memory addresses are octal values and all other numbers are decimal values. To explicitly specify a decimal number, we sometimes use a period after the last digit. To explicitly specify an octal number, we sometimes use the phrases *octal value* or *base eight*. For example, the phrase "a baker's dozen cookies" has traditionally meant 13. = 15 base eight cookies.

In this manual, AOS/VS means AOS/VS, AOS/VS II, or both, unless otherwise noted.

Finally, in examples we use

This typeface to show your entry.
This typeface to show system queries and responses.
`This typeface to show listings and status displays.`

Contacting Data General

Data General wants to assist you in any way it can to help you use its products. Please feel free to contact the company as outlined below.

Manuals

If you require additional manuals, please use the enclosed TIPS order form (United States only) or contact your local Data General sales representative.

Telephone Assistance

If you are unable to solve a problem using any manual you received with your system, free telephone assistance is available with your hardware warranty and with most Data General software service options. If you are within the United States or Canada, contact the Data General Customer Support Center (CSC) by calling 1–800–DG–HELPS. Lines are open from 8:00 a.m. to 5:00 p.m., your time, Monday through Friday. The center will put you in touch with a member of Data General's telephone assistance staff who can answer your questions.

Joining Our Users Group

Please consider joining the largest independent organization of Data General users, the North American Data General Users Group (NADGUG). In addition to making valuable contacts, members receive *FOCUS* monthly magazine, a conference discount, access to the Software Library and Electronic Bulletin Board, an annual Member Directory, Regional and Special Interest Groups, and much more. For more information about membership in the North American Data General Users Group, call 1–800–253–3902 or 1–508–443–3330.

End of Preface

Contents

Chapter 1 — Introducing the System Calls

Chapter 2 — AOS/VS, AOS/VS II, and AOS/RT32 System Calls

?A Through ?Q

086–000195 updates
093–000542

086–000195 updates
093–000542

?A Through ?Q

Index

Document Set

Chapter 2 — AOS/VS, AOS/VS II, and AOS/RT32 System Calls (Continued)

?R Through ?Z

086-000195 updates
093-000542

Appendix A — Sample Programs

Appendix B — System Log Record Format

Index

Document Set

Tables

Table **?A Through ?Q**

Figures

Figure **?A Through ?Q**

?R Through ?Z

Chapter 1
Introducing the System Calls

AOS/VS, AOS/VS II, and AOS/RT32 support a wide variety of system calls. System calls are command macros that call on predefined routines in the operating system. The system calls you code into a program allow you to

- Manage processes
- Manage logical address space
- Establish interprocess communications
- Maintain disk files
- Perform file input and output
- Manage a multitasking environment
- Define and gain access to user devices
- Establish customer/server connections between processes
- Perform input and output in blocks, rather than in records or lines
- Use the virtual timer facility (AOS/RT32 only)

You are probably familiar with the Command Line Interpreter (CLI) program. Typically it accepts statements (i.e., command lines) from a terminal, interprets them, and turns them over to the operating system for execution. An example is the command line

```
) CREATE/DIRECTORY NEW_DIR ⤶
```

Here, the CLI calls on the operating system to create a directory named NEW_DIR. More specifically, this CLI command results in an execution of the ?CREATE system call. Your programs can also call on the operating system to perform both functions that the CLI accepts (such as creating a directory file via ?CREATE) and functions that the CLI does not accept (such as getting the next name in a directory file — the function of system call ?GNFN).

Chapter 2 describes each of the AOS/VS, AOS/VS II, and AOS/RT32 system calls in alphabetical order. Also, each system call description includes a list of the error codes that are most likely to occur with the particular system call.

All memory addresses are octal and all other numbers are decimal unless otherwise specified.

We assume that you are writing assembly language programs that make system calls. However, you may write high–level language programs that also make direct calls to the operating system. In this latter case, see the general explanation later in this chapter.

In this manual, AOS/VS means AOS/VS, AOS/VS II, or both, unless otherwise noted.

Standard Format for System Calls

You must begin each system call with a question mark. Unless otherwise noted, you must reserve two return locations for each system call: a normal return (good return) and an error return (bad return). The standard format for a system call is

?CALL_NAME *[packet address]*
error return
normal return

where

[packet address]	is an optional argument to the system call macro. (See the next section, "Parameter Packets," for more explanation.) Some system calls don't have parameter packets. ?DACL is an example. Do not type the brackets; they only identify the argument as an option.
error return	is a required error return for the system call. (An error return must be a single word instruction.)
normal return	is a required normal return for the system call. (A normal return need not be a single word instruction; that is, a doubleword instruction would still be a normal return.)

Although most system calls conform to the standard format, there are exceptions. The exceptions are noted in the individual system call descriptions. An example is ?RETURN, which does not have a normal return.

When the operating system executes a system call successfully, it takes the normal return. However, if the system call fails, the operating system takes the error return. It returns an error code in AC0 that tells you why the system call failed.

Frequently, an error code does not indicate an actual error in your program, but rather indicates an exception. For example, you may want to check for error code ERFDE (File Does Not Exist), and then take another action. For simplicity, however, all exception codes are called error codes.

AC1, AC2, and AC3 represent accumulator 1, accumulator 2, and accumulator 3, respectively. The contents of AC0, AC1, and AC2 after a system call completes depend on the system call you issued. AC3, however, almost always contains the current frame pointer. There are very few references to AC3 in this manual. These references are in the descriptions of the following system calls:

- ?BRKFL
- ?IREC (Table 2–63 and Table 2–64)
- ?IXMT
- ?TASK
- ?WALKBACK

Parameter Packets

Some system calls require a *parameter packet* (or simply *packet*). A packet is a set of consecutive words that you set aside in your address space. The operating system uses these words to obtain your input specifications to a system call and/or return output values from a system call. After you set up a packet, you specify the packet address in one of the following ways.

- Load the packet's address into AC2 before you issue the system call.
- Provide the packet address as an argument to the system call macro, which will load the packet address into AC2 for you.

An individual word or doubleword in a packet is called an *offset* of the packet.

Parametric Coding

Parametric coding means writing your code using mnemonics (symbols listed in PARU.32.SR, PARU_LONG.SR, and PARU.16.SR) to refer to all error codes, packet offsets, and packet values, regardless of how the offsets are ordered in the packet figures.

You should always code parametrically because the packet figures are not true physical representations. Therefore, the exact order of the offsets may vary from one operating system revision to the next. The mnemonics, however, will always correspond to the correct packet offsets. The system call packet in Figure 1–1 is an example of parametric coding.

```
START:  ?OPEN CONSOLE      ; Issue call, with packet address "CONSOLE"
        error return       ;  as argument. Supply your own error return
        normal return      ;  and good return.

        ;Elsewhere in the program, set up the ?OPEN packet (at the
        ; address specified by "CONSOLE").

CONSOLE:.BLK   ?IBLT              ; Reserve enough words for packet
                                  ;  (?IBLT=packet length).

        .LOC    CONSOLE+?ISTI     ; Location of offset ?ISTI, the
                                  ;  file specifications word.
        .WORD   ?IEXO!?OFIN       ; Exclusive open, use as input.

        .LOC    CONSOLE+?ISTO     ; Location of offset ?ISTO, the file
                                  ;  type.
        .WORD   0                 ; Use defaults for ?ISTO.

        .LOC    CONSOLE+?IBAD     ; Location of offset ?IBAD, a byte
                                  ;  pointer to the I/O buffer.
        .DWORD  CBUFFER*2         ; CBUFFER is the console buffer.

        ;  Continue coding parametrically for the rest of the packet.
```

Figure 1–1. Parametric Coding Example

We made the packet figures, for each system call requiring a parameter packet, as close as possible to PARU.32.SR, to PARU_LONG.SR, and to PARU.16.SR (for calls with 16–bit packets). We also made the sample packets as close as possible to the packets defined in these parameter files. This arrangement helps you during debugging with the symbolic debugger utility program, since you can use the packet figure to easily match its offsets with the values displayed by the debugger.

Within some of the packet offsets (for example, those in the I/O system calls ?OPEN, ?READ, ?WRITE), you must use bit masks to select options. For example, in Figure 1–1, the specification ?IEXO!?OFIN was formed by applying the OR operator to bit masks ?IEXO and ?OFIN to select two options within the word ?ISTI. In other cases, you may need to set individual bits in a word. The notation for doublewords is

1S(bit–position)

and, provided that you have specified single words via the MASM macro .ENABLE WORD, the notation for single words is

1B(bit–position)

The S operator generates a 32–bit integer and the B operator generates a 16–bit integer. Thus, the 1S0 notation sets bit 0 (leftmost) of a 32–bit doubleword to generate 20000000000, and the 1B0 notation sets bit 0 (leftmost) of a 16–bit single word to generate 100000. Other examples are

```
.ENABLE DWORD  and  1S1   generates  10000000000
.ENABLE DWORD  and  1S2   generates  04000000000
.ENABLE DWORD  and  1S29  generates  00000000004
.ENABLE DWORD  and  1S31  generates  00000000001
.ENABLE WORD   and  1B1   generates  040000
.ENABLE WORD   and  1B2   generates  020000
.ENABLE WORD   and  1B12  generates  000010
.ENABLE WORD   and  1B15  generates  000001
```

You can also use the B operator with DWORD enabled. Another approach is to use .WORD or .DWORD with respective B or S operators. Two corresponding examples are

```
.WORD   1B15
.DWORD  1S2
```

Other System Call Input

Before you can issue many of the system calls, you must load one or more of the accumulators with input values, such as byte pointers. All accumulators are 32 bits wide under AOS/VS and AOS/RT32, as are all byte pointers.

Each system call description uses unique mnemonics for the high–order and low–order portions of 32–bit values. The high–order portion of a 32–bit value consists of the 16 most significant bits; that is, bits 0 through 15. The low–order portion of a 32–bit value consists of the 16 least significant bits; that is, bits 16 through 31.

Under AOS/VS and AOS/RT32, you must define both the high–order and the low–order portions of 32–bit doublewords. For example, if you pass –1 to indicate the default value of a 32–bit doubleword, you must set both the high–order and the low–order portions of the word to –1.

There are two methods of setting both the high– and low–order halves of a doubleword:

1. The preferred method (Method 1)

```
.LOC      CONSOLE+?IBAD
.DWORD    -1
```

This method sets both the high– and low–order bits to –1 with one easy–to–read MASM pseudo–operation.

2. The other method (Method 2)

```
.LOC      CONSOLE+?IBAD
.WORD     -1                    ; Set high-order bits to -1.

.LOC      CONSOLE+?IBAL
.WORD     -1                    ; Set low-order bits to -1.
```

This method sets the high– and low–order bits to –1 with separate MASM pseudo–operations.

There is no advantage to using Method 2. Therefore, we advise you to use Method 1. Because Method 1 is simpler than Method 2, this manual does not list low–order packet offsets. The parameter (PARU) files, especially PARU.16, do contain these offsets.

Reserved Symbols

Data General reserves all symbols that begin with a question mark (?) for its internal products, including AOS/VS and AOS/RT32. Don't *define* symbols beginning with ? because assembler errors, unexpected results, or worse can occur. A worst case scenario is your using a symbol that invokes an internal system call which, in turn, alters your program or data in a non-obvious but troublesome way. Of course, you frequently *use* DG–created symbols beginning with ? to invoke system calls and to communicate with the system calls. This manual contains hundreds of such symbols. Two of them are ?DACL, a system call name, and ?GRRH, an offset in the parameter packet for system call ?RUNTM.

A Complete Example — Assembly Language

This section describes an assembly language program that, when executed, creates directory file NEW_DIR and makes it the working directory. The program issues the system calls ?CREATE, which requires a parameter packet, and ?DIR, which does not require a parameter packet.

These system calls have the same effects as the respective CLI commands

```
) CREATE/DIRECTORY NEW_DIR ⤸
) DIR NEW_DIR ⤸
```

It's worth noting that your CLI (Command Line Interpreter) is a process executing a program named CLI.PR. The CLI accepts these two command lines, one at a time, and interprets them. Then the CLI loads accumulators, creates any necessary parameter packets, and makes the respective system calls ?CREATE and ?DIR.

The point of this section is to list the general steps of writing assembly language programs that make system calls. This section also contains a comprehensive example, based on sample program DIRCREATE.SR, from which you can easily generalize.

General Step

Unlike high–level language programs, assembly language requires no special interface to the symbols and values in SYSID.32, PARU.32, PARU_LONG, MASM_32CHAR, and PARU.16. Write

your program according to the rules of assembly language as explained in the manuals listed in the "Related Documentation" section of the Preface. While you write your program, code its system calls according to their documentation in this manual.

Program Listing

Figure 1–2 contains the listing file DIRCREATE.LS that the Macroassembler created from source program file DIRCREATE.SR.

```
SOURCE: DIRCREATE        MASM 06.00.00.00     27-AUG-85 10:12:04  PAGE  1
01                                .TITLE DIRCREATE
02                                .ENT   DIRCREATE
03                                .NREL
04
05                                DIRCREATE:
06 000000 UC 122071 000100        XLEFB  0,NEWDIR*2  ; Byte pointer
07                                                   ;   to new
08                                                   ;   directory's
                                                     ;   name
09 000002 UC 124531               WSUB   1,1         ; 0 in AC1
10                                ?CREATE DIRPKT     ; Create the direc-
11                                                   ;   tory according
12                                                   ;   to its parameter
13                                                   ;   packet.
14 000011 UC 105470               WBR     ERROR      ; Error return
15                                ; Make newly created directory
16                                ;   NEW_DIR the default one.
17 000012 UC 122071 000054        XLEFB  0,NEWDIR*2  ; Byte pointer
18                                                   ;   to new direc-
19                                                   ;   tory's name
20 000014 UC 124531               WSUB    1,1    ; 0 in AC1
21 000015 UC 150531               WSUB    2,2    ; 0 in AC2
22                                ?DIR             ; Move to NEW_DIR.
23 000021 UC 104470               WBR     ERROR
24 000022 UC 150531               WSUB    2,2    ; Execute a normal
25                                ?RETURN          ;   return to AOS/VS.
26 000026 UC 101770               WBR     ERROR
27
28                                DIRPKT:
29 000027 UC  00000000012         .BLK    ?CLTH  ; Packet length
30             00000000027  UC    .LOC    DIRPKT+?CFTYP
31 000027 UC 000012               .WORD   0*400+?FDIR ; Left byte is 0,
32                                                    ;       right specifies
33                                                    ;       a standard (non-
34                                                    ;       control point)
35                                                    ;       directory.
36              00000000030  UC   .LOC    DIRPKT+?CHFS
37 000030 UC 177777               .WORD   -1     ; Hashframe size is
38                                                ;   the default.
```

Figure 1–2. Listing File of Program DIRCREATE.SR (continued)

```
   SOURCE: DIRCREATE     MASM 06.00.00.00    27-AUG-85 10:12:04  PAGE  2
01                00000000031  UC    .LOC    DIRPKT+?CTIM
02 000031 UC      37777777777        .DWORD  -1      ; Time block is cur-
03                                                   ;   rent date, time.
04                00000000033  UC    .LOC    DIRPKT+?CACP
05 000033 UC      37777777777        .DWORD  -1      ; ACL same as caller's
06                00000000035  UC    .LOC    DIRPKT+?CMSH
07 000035 UC      00000000000        .DWORD  0       ; Set to zero for
08                                                   ;   type ?FDIR.
09                00000000037  UC    .LOC    DIRPKT+?CMIL
10 000037 UC      177777             .WORD   -1      ; Default number of
11                                                   ;   index levels.
12                00000000040  UC    .LOC    DIRPKT+?CMRS
13 000040 UC      000000             .WORD   0       ; Reserved.
14                00000000041  UC    .LOC    DIRPKT+?CLTH ; End of packet
15
16 000041 UC      047105 053537      NEWDIR: .TXT "NEW_DIR" ; Directory name
17                042111 051000
18
19 000045 UC      153211             ERROR:  WLDAI   ?RFEC+?RFCF+?RFER,2 ;Error
20                 00000150000
21                                                                      ;return
22                                   ?RETURN
23 000053 UC      175270             WBR     ERROR
24
25                                   .END    DIRCREATE

-----------------

XREF:   DIRCREATE       MASM 06.00.00.00     27-AUG-85 10:12:04  PAGE  3
?CACP     00000000004       2/04
?CFTYP    00000000000       1/30
?CHFS     00000000001       1/36
?CLTH     00000000012       1/29    2/14
?CMIL     00000000010       2/09
?CMRS     00000000011       2/12
?CMSH     00000000006       2/06
?CREATE   00000000000 MA    1/10
?CTIM     00000000002       2/01
?DIR      00000000000 MA    1/22
?FDIR     00000000012       1/31
?RETURN   00000000000 MA    1/25    2/22
?RFCF     00000100000       2/19
?RFEC     00000010000       2/19
?RFER     00000040000       2/19
?SYST     00000000000 MA    1/11    1/23    1/26    2/23
?XCALL    00000000001       1/11    1/11    1/23    1/23    1/26    1/26
                            2/23    2/23
DIRCREAT  00000000000 EN    1/02    1/05#   2/25
DIRPKT    00000000027       1/11    1/28#   1/30    1/36    2/01    2/04
                            2/06    2/09    2/12    2/14
ERROR     00000000045       1/14    1/23    1/26    2/19#   2/23
NEWDIR    00000000041       1/06    1/17    2/16#

NO ASSEMBLY ERRORS
```

Figure 1–2. Listing File of Program DIRCREATE.SR (concluded)

Program Construction

First, be sure you have at least Read access to files MASM.PR, MASM.PS, and LINK.PR. File MASM.PR is the Macroassembler. MASM.PS is a MASM.PR–created file that includes SYSID.32.SR, PARU.32.SR, and PARU_LONG.SR in a special format. Typically, files MASM.PR, MASM.PS, and LINK.PR are in directory :UTIL.

Next are the commands that show the construction of program file DIRCREATE.PR from source file DIRCREATE.SR. All filename suffixes appear for emphasis; you frequently can ignore them during commands to the Macroassembler and Link. The output from the Macroassembler and Link programs doesn't occur in the following commands.

```
) DELETE/2=IGNORE DIRCREATE.LS ↵
) COMMENT Create object file DIRCREATE.OB and list file ↵
) COMMENT    DIRCREATE.LS from DIRCREATE.SR. ↵
) XEQ MASM/L=DIRCREATE.LS DIRCREATE.SR ↵
) COMMENT Create program file DIRCREATE.PR from object file DIRCREATE.OB. ↵
) XEQ LINK DIRCREATE.OB ↵
) COMMENT Program file DIRCREATE.PR is now ready for execution. ↵
```

Program Execution

Next is the dialog that shows the execution of program file DIRCREATE.PR. Read it carefully to see what happens when directory file NEW_DIR does not exist prior to executing DIRCREATE.PR, and then what happens when NEW_DIR does exist prior to executing DIRCREATE.PR.

```
) COMMENT EXECUTE DIRCREATE.PR after making sure that NEW_DIR doesn't exist. ↵
) DELETE/2=IGNORE NEW_DIR ↵
) DIRECTORY ↵
UDD4:ALICE
) DATE; TIME ↵
27-AUG-85
10:14:40
) XEQ DIRCREATE.PR ↵


) COMMENT There were no errors when ?CREATE and ?DIR executed, so no errors ↵
) COMMENT    were reported. ↵
) FILESTATUS/ASSORT NEW_DIR ↵


DIRECTORY :UDD4:ALICE


 NEW_DIR        DIR   27-AUG-85   10:14:48       0
```

```
) DIRECTORY ↵
:UDD4:ALICE
) COMMENT The ?DIR call in DIRCREATE.PR executed successfully, but ↵
) COMMENT    NEW_DIR was the current directory only during the rest ↵
) COMMENT    of the life of process DIRCREATE.PR. When the process ↵
) COMMENT    terminated, the directory, :UDD4:ALICE, of the parent ↵
) COMMENT    process once again became the current directory. ↵
) DIRECTORY NEW_DIR ↵

) FILESTATUS ↵

) COMMENT As expected, the FILESTATUS command returns no information ↵
) COMMENT    because directory NEW_DIR is empty. ↵
) DIRECTORY ^ ↵
) COMMENT We're back in directory ALICE. Execute DIRCREATE.PR ↵
) COMMENT    again, and see what happens when ?CREATE tries to ↵
) COMMENT    create a file (NEW_DIR) that already exists. ↵
) XEQ DIRCREATE.PR ↵


*ERROR*
FILE NAME ALREADY EXISTS
ERROR: FROM PROGRAM
XEQ,DIRCREATE.PR


) COMMENT Notice how ?RETURN reported the error and terminated ↵
) COMMENT    the process. ↵
```

Error Codes

When a system call takes an error return, it places an error code in AC0. Each error code has an octal value and a mnemonic that represents the code. In addition, each error code has a text message associated with it.

You will find the manual *AOS/VS II Error and Status Messages* useful as you read the error codes.

The error codes whose format is ERxxx come from file PARU.32.SR. The error codes that begin with "ER_" come from file MASM_32CHAR.PS.

High–Level Language Interface

The previous section explained how assembly language programs can interface with the operating system. High–level language programs can also make system calls. These high–level languages are

- Ada
- BASIC
- C
- COBOL
- FORTRAN 77
- Pascal
- PL/I

The principle for writing code that makes system calls is the same for each of these languages. For any system call in any language, follow these four steps:

1. Read about the system call in this manual. Note the input/output values of the accumulators and, if any, the parameter packet.

2. Read the language's documentation for its technique for making system calls.

3. Assign values to accumulator variables and to, if any, an array that becomes the parameter packet.

   NOTE: In a high–level language statement where you define an ASCII character string that will be assigned to an accumulator, you must terminate the string with a null character. If the string does not end with a NL, FF, CR or null character, an error occurs.

   For example,

   ```
   CHARACTER*8 NEW_DIR / 'TEST_DIR<0>' /
   AC0 = BYTEADDR(NEW_DIR)
   ```

   The first FORTRAN statement defines the NEW_DIR byte pointer and the text string, TEST_DIR, which terminates with a null character. The second statement loads the string into AC0.

4. Write the statement that makes the system call. The statement includes or refers to the accumulator variables.

For example, suppose you want to create a FORTRAN 77 program that does the same things as assembly language program DIRCREATE.SR. The corresponding four steps in the creation of DIRCREATE.F77 would be as follows.

1. Read the documentation of ?CREATE (with emphasis on the directory option) and ?DIR in this system call dictionary.

2. Read the chapter about the system interface function, ISYS, in the *FORTRAN 77 Environment Manual (AOS/VS).*

3. Use 4–byte integer variables for the values of AC0, AC1, and AC2 when preparing for ?CREATE (whose FORTRAN 77 counter part is most likely 4–byte integer variable

ISYS_CREATE) and for ?DIR (correspondingly ISYS_DIR). Create an array of 2–byte integer variables that is the parameter packet for ?CREATE. Use the BYTEADDR and WORDADDR functions as needed for byte addresses and word addresses, respectively.

4. Use the ISYS function to make the system call. The four arguments to this function are the system call identifier, contents of AC0, contents of AC1, and contents of AC2. The result that ISYS returns is either 0 for no error, or the error code that the operating system placed in AC0.

Appendix A contains FORTRAN 77 program DIRCREATE.F77 that has the same functionality as assembly language program DIRCREATE.SR.

End of Chapter

Chapter 2
AOS/VS, AOS/VS II, and AOS/RT32 System Calls

This chapter contains the system calls for all three operating systems. The calls appear in alphabetical order.

Most calls apply equally to the operating systems. The classes of exceptions to this statement are

- System calls that apply to AOS/VS. An example is ?JPSTAT.
- System calls that apply to AOS/RT32 only. An example is ?FIDEF.
- System calls that behave differently on the operating systems. An example is ?PSTAT.

In each of these three cases we indicate that operating system differences occur. In the third case we list the specific differences wherever they occur — accumulators, parameter packet, etc.

The categories of information we supply for each system call follow.

- Name and function.
- Summary of operating system differences (if any).
- Syntax of the assembly language statement call, including error and normal returns.
- Accumulator input and output.
- Error codes.
- Why you might use the call.
- Who can use (i.e., issue) the call, in terms of required process privileges and file access rights.

  NOTE: This category applies only to AOS/VS. Under AOS/RT32, all processes have all process privileges and all files have the same access control list: +,OWARE.

- What the call does.
- A figure to show the structure of the parameter packet (if any); this figure is helpful while you are using the debugger and need to examine/change the contents of the packet.
- A table to show the contents of the parameter packet (if any); this table is helpful while you are preparing the contents of the packet during the coding step of program development.
- Notes and references to related documentation (if any). The references are only to other parts of this manual.
- Sample packet (if any).

Also, you will find the sample programs in Appendix A helpful as you code the system calls in your programs. Use this manual's index to look for sample programs that issue specific system calls. For example, the index entries for system call ?GTMES include pointers to sample programs BOOMER.SR, DLIST.SR, and TIMEOUT.SR. The index entries for these three sample programs contain their actual page numbers.

Remember — this manual does not explain software concepts, such as the definition of a directory and how you use the CLI program to create and move to directories. Read the *AOS/VS System Concepts* Manual and *Introduction to AOS/RT32* for general information about the functionality of system calls. Read this manual for specific information about how to implement these system calls from your programs.

Summary Table

Table 2–1 summarizes all AOS/VS, AOS/VS II, and AOS/RT32 system calls, and groups them according to their functions. It also indicates operating system differences. The functions, which appear in Table 2–1 in the following order, are

- AOS/RT32 system calls
- Memory management
- Process management
- File creation and management
- File input/output
- Debugging
- Windowing
- Multitasking
- Interprocess communications
- Connection management
- Multiprocessor management
- Class scheduling
- AOS/VS system resources
- User devices
- Bisynchronous communications
- 16–bit processes

There are six columns of information in Table 2–1. Their contents are

- System call name.
- System call description.
- Y to indicate that AOS/VS supports the system call, or nothing to indicate otherwise.
- Y to indicate that AOS/VS II supports the system call, or nothing to indicate otherwise.
- Y to indicate that AOS/RT32 supports the system call, or nothing to indicate otherwise.
- Y to indicate there is a difference in functionality or coding between AOS/VS, AOS/VS II and AOS/RT32 or nothing to indicate otherwise. As explained earlier in the chapter, we indicate the specific differences wherever they occur — accumulator, parameter packet, etc.

Table 2–1. Summary of AOS/VS and AOS/RT32 System Calls

System Call	Function: AOS/RT32 System Calls Description	AOS/VS	AOS/VSII	AOS/RT32	OS Diff
?FIDEF	Defines a fast user device.			Y	
?FIXMT	Transmits a message from an interrupt service routine within Ring 0.			Y	
?VMEM	Changes the partition size of a process.			Y	
?VTFCREATE	Creates a Virtual Timer Facility timer.			Y	
?VTFKILL	Kills a Virtual Timer Facility timer.			Y	
?VTFMODIFY	Modifies a Virtual Timer Facility timer.			Y	
?VTFSUS	Suspends or restarts a Virtual Timer Facility Timer.			Y	
?VTFXIT	Exits from a Virtual Timer Facility interrupt routine.			Y	
?WTVERR	Waits for a Virtual Timer Facility error message.			Y	
?WTVSIG	Waits for a Virtual Timer Facility signal.			Y	
	Function: Memory Management				
System Call	Description	AOS/VS	AOS/VSII	AOS/RT32	OS Diff
?ESFF	Flushes shared file memory pages to disk	Y	Y	Y	Y
?FLUSH	Flushes the contents of a shared page to disk.	Y	Y	Y	
?GMEM	Returns the number of undedicated memory pages.	Y	Y	Y	
?GSHPT	Lists the current shared partition size.	Y	Y	Y	
?LMAP	Maps a lower ring.	Y	Y	Y	
?MEM	Lists the current unshared memory parameters.	Y	Y	Y	
?MEMI	Changes the number of unshared pages in the logical address space.	Y	Y	Y	
?PMTPF	Permits access to a protected file.	Y	Y	Y	
?RPAGE	Releases a shared page.	Y	Y	Y	
?SCLOSE	Closes a file previously opened for shared access.	Y	Y	Y	
?SOPEN	Opens a file for shared access.	Y	Y	Y	
?SOPPF	Opens a protected shared file.	Y	Y	Y	
?SPAGE	Performs a shared-page read.	Y	Y	Y	
?SSHPT	Establishes a new shared partition.	Y	Y	Y	
?UNWIND	Unwinds the stack and restores the previous environment.	Y	Y	Y	
?VALAD	Validates a logical address.	Y	Y	Y	
?VALIDATE	Validates an area for Read or Write access.	Y	Y	Y	

(continued)

Table 2–1. Summary of AOS/VS and AOS/RT32 System Calls

System Call	Function: Process Management (continued) Description	AOS/VS	AOS/VSII	AOS/RT32	OS Diff
?AWIRE	Changes the wiring characteristics of the Agent.	Y	Y		
?BLKPR	Blocks a process.	Y	Y	Y	
?BRKFL	Terminates a process and creates a break file.	Y	Y		
?CHAIN	Passes control from a Ring 7 caller to a new program.	Y	Y	Y	
?CTYPE	Changes a process type.	Y	Y	Y	
?DADID	Gets the PID of a process's father.	Y	Y	Y	
?ENBRK	Enables a break file.	Y	Y		
?EXPO	Sets, clears, or examines execute-protection status.	Y	Y	Y	
?GBIAS	Gets the current bias factor values.	Y	Y	Y	
?GPID	Returns all active PIDs based on a host ID.	Y	Y	Y	
?GUNM	Gets the username of a process.	Y	Y	Y	
?GVPID	Gets the virtual PID of a process.	Y	Y		
?HNAME	Gets a hostname or host identifier.	Y	Y	Y	
?KHIST	Kills a histogram.	Y	Y	Y	
?LOCALITY	Changes user locality.	Y	Y		
?MDUMP	Dumps the memory image from a user-specified ring to a file.	Y	Y	Y	
?MPHIST	Starts a histogram on a uni- or multiprocessor system.	Y	Y		
?PCLASS	Get a process's class and locality.	Y	Y		
?PIDS	Gets information about PIDs.	Y	Y	Y	
?PNAME	Gets a full process name.	Y	Y	Y	
?PRIPR	Changes the priority of a process.	Y	Y	Y	
?PROC	Creates a process.	Y	Y	Y	Y
?PROFILE	Performs a profile request.	Y	Y		
?PSTAT	Returns status information on a process.	Y	Y	Y	Y
?RESCHED	Reschedules current time slice.	Y	Y	Y	Y

(continued)

Table 2–1. Summary of AOS/VS and AOS/RT32 System Calls

System Call	Function: Process Management (concluded) Description	AOS/VS	AOS/VSII	AOS/RT32	OS Diff
?RETURN	Terminates the calling process; passes the termination message to the father.	Y	Y	Y	
?RINGLD	Loads a program file into a specified ring.	Y	Y	Y	
?RNGPR	Returns the .PR filename for a ring.	Y	Y	Y	
?RNGST	Stops ?RINGLD from loading lower rings.	Y	Y	Y	
?RUNTM	Gets runtime statistics on a process.	Y	Y	Y	
?SBIAS	Sets the bias factors.	Y	Y		
?SONS	Gets a list of son processes for a target PID.	Y	Y	Y	
?SUPROC	Enters, leaves, or examines Superprocess mode.	Y	Y	Y	Y
?SUSER	Enters, leaves, or examines Superuser mode.	Y	Y	Y	Y
?SYSPRV	Enters, leaves, or examines a process state.	Y	Y	Y	Y
?TERM	Terminates a process.	Y	Y	Y	
?TPID	Translates a PID.	Y	Y	Y	
?UBLPR	Unblocks a process.	Y	Y	Y	
?UNWIRE	Unwires pages previously wired.	Y	Y	Y	
?WHIST	Starts a histogram.	Y	Y	Y	
?WIRE	Wires pages to the working set.	Y	Y	Y	
?XPSTAT	Returns extended status information on a process.	Y	Y	Y	Y

(continued)

Table 2–1. Summary of AOS/VS and AOS/RT32 System Calls

System Call	Function: File Creation and Management Description	AOS/VS	AOS/VSII	AOS/RT32	OS Diff
?CGNAM	Gets a complete pathname from a channel number.	Y	Y	Y	
?CKVOL	Checks volume identifier of a labeled magnetic tape.	Y	Y		
?CPMAX	Sets maximum size for a control point directory.	Y	Y	Y	Y
?CREATE	Creates a file or directory.	Y	Y	Y	Y
?DACL	Sets, clears, or examines a default access control list.	Y	Y	Y	
?DELETE	Deletes a file entry.	Y	Y	Y	Y
?DIR	Changes the working directory.	Y	Y	Y	
?FLOCK	Locks an object.	Y	Y		
?FSTAT	Gets file status information.	Y	Y	Y	Y
?FUNLOCK	Unlocks an object.	Y	Y		
?GACL	Gets a file entry's access control list.	Y	Y	Y	Y
?GLINK	Gets the contents of a link entry.	Y	Y	Y	
?GLIST	Gets the contents of a search list.	Y	Y	Y	
?GNAME	Gets a complete pathname.	Y	Y	Y	Y
?GNFN	Lists a particular directory's entries.	Y	Y	Y	
?GPRNM	Gets a program's pathname.	Y	Y	Y	
?GRNAME	Returns complete pathname of generic file.	Y	Y	Y	
?GROUP	Changes group ACL list.		Y		
?GTACP	Gets access control privileges.	Y	Y	Y	Y
?INIT	Initializes a logical disk.	Y	Y		
?LDUINFO	Obtains logical disk information.		Y		
?MIRROR	Mirrors and synchronizes LDU images.	Y	Y		Y
?RECREATE	Recreates a file.	Y	Y	Y	
?RELEASE	Releases an initialized logical disk.	Y	Y	Y	
?RENAME	Renames a file.	Y	Y	Y	
?RNAME	Determines whether a pathname contains a reference to a remote host.	Y	Y	Y	
?SACL	Sets a new access control list.	Y	Y	Y	Y
?SATR	Sets or removes the Permanence attribute for a file or directory.	Y	Y	Y	
?SLIST	Sets the search list for the calling process.	Y	Y	Y	
?SYLOG	Manipulates the system log file.	Y	Y		
?XCREATE	Creates a file or directory (extended).		Y		
?XFSTAT	Gets file status information (extended).		Y		
?XGTACP	Gets ACL privileges (extended).		Y		
?XINIT	Initializes a logical disk (extended).	Y	Y		Y

(continued)

Table 2–1. Summary of AOS/VS and AOS/RT32 System Calls

System Call	Function: File Input/Output (continued) Description	AOS/VS	AOS/VSII	AOS/RT32	OS Diff
?ALLOCATE	Allocates disk blocks.	Y	Y	Y	Y
?ASSIGN	Assigns a character device to a process.	Y	Y		
?BLKIO	Performs (reads/writes) block I/O.	Y	Y	Y	Y
?CLOSE	Closes an open channel.	Y	Y	Y	
?CRUDA	Creates a user data area.	Y	Y	Y	Y
?DEASSIGN	Deassigns a character device.	Y	Y	Y	
?FEOV	Forces end-of-volume on labeled magnetic tape.	Y	Y		
?GCHR	Reads device characteristics of a character device.	Y	Y	Y	
?GCLOSE	Closes a file previously opened for block I/O.	Y	Y	Y	
?GDLM	Gets a delimiter table.	Y	Y	Y	
?GECHR	Get extended characteristics.	Y	Y	Y	
?GOPEN	Opens a file for block I/O.	Y	Y	Y	Y
?GPOS	Gets the current file-pointer position.	Y	Y	Y	
?GTRUNCATE	Truncates a disk file.	Y	Y	Y	Y
?INTWT	Defines a terminal interrupt task.	Y	Y	Y	
?KINTR	Simulates keyboard interrupt sequences.	Y	Y	Y	
?KIOFF	Disables control-character terminal interrupts.	Y	Y	Y	
?KION	Re-enables control-character terminal interrupts.	Y	Y	Y	
?KWAIT	Waits for a terminal interrupt.	Y	Y	Y	
?LABEL	Creates a label for a magnetic tape.	Y	Y		
?ODIS	Disables terminal interrupts.	Y	Y	Y	
?OEBL	Enables terminal interrupts.	Y	Y	Y	
?OPEN	Opens a file.	Y	Y	Y	
?PRDB	Performs physical block Reads.	Y	Y	Y	
?PWRB	Performs physical block Writes.	Y	Y	Y	Y
?RDB/?WRB	Performs (reads/writes) block I/O.	Y	Y	Y	Y
?RDUDA/ ?WRUDA	Reads/writes a user data area.	Y	Y	Y	Y
?READ/ ?WRITE	Performs (reads/writes) record I/O.	Y	Y	Y	

(continued)

Table 2–1. Summary of AOS/VS and AOS/RT32 System Calls

System Call	Function: File Input/Output (concluded) Description	AOS/VS	AOS/VSII	AOS/RT32	OS Diff
?SCHR	Sets a character device's characteristics.	Y	Y	Y	
?SDLM	Sets a delimiter table.	Y	Y	Y	
?SECHR	Set extended characteristics of a device.	Y	Y	Y	
?SEND	Sends a message to a terminal.	Y	Y	Y	Y
?SPOS	Sets the position of the file pointer.	Y	Y	Y	
?STOM	Sets the time-out value for a device.	Y	Y	Y	Y
?TRUNCATE	Truncates a file at the current position.	Y	Y	Y	
?UPDATE	Flushes file descriptor information.	Y	Y	Y	Y
?WRB	Writes block I/O.	Y	Y	Y	Y
?WRITE	Writes record I/O.	Y	Y	Y	
?WRUDA	Writes a user data area.	Y	Y	Y	Y

System Call	Function: Debugging Description	AOS/VS	AOS/VSII	AOS/RT32	OS Diff
?DEBUG	Calls the Debugger utility.	Y	Y	Y	
?GTNAM	Returns symbol closest in value to specified input value.	Y	Y		
?GTSVL	Gets the value of a user symbol.	Y	Y		

(continued)

Table 2–1. Summary of AOS/VS and AOS/RT32 System Calls

System Call	Function: Windowing Description	AOS/VS	AOS/VSII	AOS/RT32	OS Diff
?GRAPHICS	Manipulates pixel maps.	Y	Y		
?PTRDEVICE	Controls input from a pointer device.	Y	Y		
?WINDOW	Manipulates windows.	Y	Y		
System Call	Function: Multitasking (continued) Description	AOS/VS	AOS/VSII	AOS/RT32	OS Diff
?DFRSCH	Disables task rescheduling and indicates prior state of rescheduling.	Y	Y	Y	
?DQTSK	Dequeues one or more previously queued tasks.	Y	Y	Y	
?DRSCH	Disables scheduling.	Y	Y	Y	
?ERSCH	Enables multitask scheduling for the calling process.	Y	Y	Y	
?IDGOTO	Redirects a task's execution path.	Y	Y	Y	
?IDKIL	Kills a task specified by its TID.	Y	Y	Y	
?IDPRI	Changes the priority of a task specified by its TID.	Y	Y	Y	
?IDRDY	Readies a task specified by its TID.	Y	Y	Y	
?IDSUS	Suspends a task specified by its TID.	Y	Y	Y	
?IFPU	Initializes the floating-point unit.	Y	Y	Y	
?IQTSK	Creates a queued task manager.	Y	Y	Y	
?KILAD	Defines a kill-processing routine.	Y	Y	Y	
?KILL	Kills the calling task.	Y	Y	Y	
?MYTID	Gets the priority and TID of the calling task.	Y	Y	Y	
?PRI	Changes the priority of the calling task.	Y	Y	Y	

(continued)

Table 2–1. Summary of AOS/VS and AOS/RT32 System Calls

System Call	Function: Multitasking (concluded) Description	AOS/VS	AOS/VSII	AOS/RT32	OS Diff
?PRKIL	Kills all tasks of a specified priority.	Y	Y	Y	
?PRRDY	Readies all tasks of a specified priority.	Y	Y	Y	
?PRSUS	Suspends all tasks of a specified priority.	Y	Y	Y	
?REC	Receives an intertask message.	Y	Y	Y	
?RECNW	Receives an intertask message without waiting.	Y	Y	Y	
?SUS	Suspends the calling task.	Y	Y	Y	
?TASK	Initiates one or more tasks.	Y	Y	Y	
?TIDSTAT	Returns status of target task.	Y	Y	Y	
?TLOCK	Protects a task from being redirected.	Y	Y	Y	
?TRCON	Reads a task message from the process terminal.	Y	Y		
?TUNLOCK	Allows a task to be redirected.	Y	Y	Y	
?UIDSTAT	Returns the status of a task and an unambiguous identifier.	Y	Y	Y	
?WDELAY	Suspends a task for a specified time.	Y	Y	Y	
?XMT	Transmits an intertask message.	Y	Y	Y	
?XMTW	Transmits an intertask message and waits for it to be received.	Y	Y	Y	

System Call	Function: Interprocess Communications Description	AOS/VS	AOS/VSII	AOS/RT32	OS Diff
?GCPN	Gets the terminal port number.	Y	Y		
?GPORT	Returns the PID associated with a global port number.	Y	Y	Y	
?ILKUP	Returns a global port number.	Y	Y	Y	
?IMERGE	Modifies a ring field within a global port number.	Y	Y	Y	
?IREC	Receives an IPC message.	Y	Y	Y	
?ISEND	Sends an IPC message.	Y	Y	Y	
?ISPLIT	Finds the owner of a port (including its ring number).	Y	Y	Y	
?IS.R	Sends, and then receives an IPC message.	Y	Y	Y	
?TMSG	Defines the termination message format.	Y	Y	Y	
?TPORT	Translates a local port number to its global equivalent.	Y	Y	Y	

(continued)

Table 2–1. Summary of AOS/VS and AOS/RT32 System Calls

System Call	Description	AOS/VS	AOS/VSII	AOS/RT32	OS Diff
	Function: Connection Management				
?CON	Becomes a customer of a specified server.	Y	Y	Y	
?CTERM	Terminates a customer process.	Y	Y	Y	
?DCON	Breaks a connection (disconnects) in Ring 7.	Y	Y	Y	
?DRCON	Breaks a connection (disconnects).	Y	Y	Y	
?MBFC	Moves bytes from a customer's buffer.	Y	Y	Y	
?MBTC	Moves bytes to a customer's buffer.	Y	Y	Y	
?PCNX	Passes a connection from one server to another in Ring 7.	Y	Y	Y	
?PRCNX	Passes a connection from one server to another.	Y	Y	Y	
?RESIGN	Resigns as a server.	Y	Y	Y	
?SERVE	Becomes a server.	Y	Y	Y	
?SIGNL	Signals another task.	Y	Y	Y	
?SIGWT	Signals another task, and then waits for a signal.	Y	Y	Y	
?VCUST	Verifies a customer in Ring 7.	Y	Y	Y	
?VRCUST	Verifies a customer in a specified ring.	Y	Y	Y	
?WTSIG	Waits for a signal from another task or process.	Y	Y	Y	
	Function: Multiprocessor Management				
?JPINIT	Initializes a job processor.	Y	Y		
?JPMOV	Moves a job processor to a new logical processor.	Y	Y		
?JPREL	Releases a job processor.	Y	Y		
?JPSTAT	Gets the status of a job processor.	Y	Y		

(continued)

Table 2–1. Summary of AOS/VS and AOS/RT32 System Calls

System Call	Function: Class Scheduling Description	AOS/VS	AOS/VSII	AOS/RT32	OS Diff
?CLASS	Gets or sets class IDs.	Y	Y		
?CLSCHED	Enables, disables, or examines class scheduling.	Y	Y		
?CLSTAT	Returns class scheduling statistics.	Y	Y		
?CMATRIX	Gets or sets the class matrix.	Y	Y		
?LPCLASS	Gets/sets logical processor class assignments.	Y	Y		
?LPCREA	Creates a logical processor.	Y	Y		
?LPDELE	Deletes a logical processor.	Y	Y		
?LPSTAT	Gets the status of a logical processor.	Y	Y		
	Function: AOS/VS System Resources				
System Call	**Description**	**AOS/VS**	**AOS/VSII**	**AOS/RT32**	**OS Diff**
?BNAME	Determines whether process name/queue name is on local or remote host.	Y	Y	Y	
?CDAY	Converts a scalar date value.	Y	Y	Y	
?CONFIG	Displays or resets the current message-based reliable (MRC) device routes.		Y		
?CONINFO	Requests addressing information on a terminal or a console.		Y		
?CTOD	Converts a scalar time value.	Y	Y	Y	
?ENQUE	Sends a message to IPC and spooler files	Y	Y	Y	
?ERMSG	Reads the error message file.	Y	Y	Y	
?EXEC	Requests a service from EXEC.	Y	Y		
?FDAY	Converts date to a scalar value.	Y	Y	Y	
?FEDFUNC	Interfaces to File Editor utility.	Y	Y	Y	
?FTOD	Converts time of day to a scalar value.	Y	Y	Y	
?GDAY	Gets the current date.	Y	Y	Y	
?GHRZ	Gets the frequency of the system clock.	Y	Y	Y	
?GSID	Gets the system identifier.	Y	Y	Y	
?GTIME	Gets the time, date, and time zone.	Y	Y		
?GTMES	Gets an initial IPC message.	Y	Y	Y	
?GTOD	Gets the time of day.	Y	Y	Y	
?GUHPI	Gets unique hardware processor ID.	Y	Y		
?ITIME	Returns the OS-format internal time.	Y	Y	Y	
?LOGCALLS	Logs system calls.	Y	Y	Y	
?LOGEV	Enters an event in the system log file.	Y	Y		
?NTIME	Sets the time, date, and time zone.	Y	Y		
?OPER	Creates and maintains an operator interface.	Y	Y		
?OPEX	Communicates between the current process and an operator process.	Y	Y		
?PWDCRYP	Performs a password data encryption request.	Y	Y		
?RTODC	Reads the time-of-day conversion data.	Y	Y		

(continued)

Table 2–1. Summary of AOS/VS and AOS/RT32 System Calls

System Call	Function: AOS/VS System Resources Description	AOS/VS	AOS/VSII	AOS/RT32	OS Diff
?SDAY	Sets the system calendar.	Y	Y	Y	
?SINFO	Gets selected information about the current operating system.	Y	Y	Y	Y
?SSID	Sets the system identifier.	Y	Y	Y	
?STOD	Sets the system clock.	Y	Y	Y	
	Function: User Devices				
System Call	**Description**	**AOS/VS**	**AOS/VSII**	**AOS/RT32**	**OS Diff**
?CLRDV	Clears a device.	Y	Y	Y	
?DDIS	Disables access to all devices.	Y	Y	Y	
?DEBL	Enables access to all devices.	Y	Y	Y	
?IDEF	Defines a user device.	Y	Y	Y	
?IMSG	Receives an interrupt service message.	Y	Y	Y	
?IRMV	Removes a user device.	Y	Y	Y	
?IXIT	Exits from an interrupt service routine.	Y	Y	Y	
?IXMT	Transmits a message from an interrupt service routine.	Y	Y	Y	
?LEFD	Disables LEF mode.	Y	Y	Y	
?LEFE	Enables LEF mode.	Y	Y	Y	
?LEFS	Returns the current LEF mode status.	Y	Y	Y	
?MAPDV	Maps a device into logical address space.	Y	Y	Y	Y
?STMAP	Sets the data channel map.	Y	Y	Y	
	Function: Bisynchronous Communications				
System Call	**Description**	**AOS/VS**	**AOS/VSII**	**AOS/RT32**	**OS Diff**
?SDBL	Disables a BSC line.	Y	Y	Y	
?SDPOL	Defines a polling list or a poll-address/select-address pair.	Y	Y	Y	
?SDRT/ ?SERT	Disables/re-enables a relative terminal.	Y	Y	Y	
?SEBL	Enables a BSC line.	Y	Y	Y	
?SERT	Re-enables a relative terminal.	Y	Y	Y	
?SGES	Gets BSC error statistics.	Y	Y	Y	
?SRCV	Receives data or a control sequence over a BSC line.	Y	Y	Y	
?SSND	Sends data or a control sequence over a BSC line.	Y	Y	Y	

(continued)

Table 2–1. Summary of AOS/VS and AOS/RT32 System Calls

System Call	Function: 16-bit Processes Description	AOS/VS	AOS/VSII	AOS/RT32	OS Diff
?DELAY	Suspends a 16-bit task for a specified interval.	Y	Y	Y	
?GCRB	Gets the base of the current resource.	Y	Y	Y	
?IDSTAT	Returns task status word.	Y	Y	Y	
?IESS	Initializes an extended state save area.	Y	Y	Y	
?IHIST	Starts a histogram for a 16-bit process.	Y	Y	Y	
?KCALL	Keeps the calling resource and acquires a new resource.	Y	Y	Y	
?OVEX	Releases an overlay and returns.	Y	Y	Y	
?OVKIL	Exits from an overlay and kills the calling task.	Y	Y	Y	
?OVLOD	Loads and goes to an overlay.	Y	Y	Y	
?OVREL	Releases an overlay area.	Y	Y	Y	
?RCALL	Releases one resource and acquires a new one.	Y	Y	Y	
?RCHAIN	Chains to a new procedure.	Y	Y	Y	
?SERMSG	Returns text for associated error code.	Y	Y	Y	
?WALKBACK	Returns information about previous frames in the stack.	Y	Y	Y	

(concluded)

?ALLOCATE *?RDB/?WRB packet address*
error return
normal return

Input

AC0	Reserved (Set to 0.)
AC1	Channel number
AC2	Address of the ?RDB/?WRB

Output

AC0	Undefined
AC1	Number of bytes allocated (number of blocks * 512)
AC2	Unchanged packet

Error Codes in AC0

ERCPD	Control point directory maximum exceeded
ERFNO	Channel not open
ERICB	Insufficient contiguous disk blocks
ERSIM	Simultaneous requests on same channel
ERVWP	Invalid word pointer passed as a system call argument
ERWAD	Write access denied
ER_FS_DIRECTORY_NOT_AVAILABLE	Directory not available because the LDU was force released (AOS/VS II only)
ER_FS_TLA_MODIFY_VIOLATION	Attempt to modify an AOS/VS II file with ?ODTL value supplied in ?GOPEN packet

File system error codes

Why Use It?

?ALLOCATE makes sure that subsequent I/O will not cause a calling process to exceed its control point directory's maximums.

Who Can Use It?

There are no special process privileges needed to issue this call. You must have obtained a channel number to the file, via ?GOPEN or ?SOPEN, before issuing ?ALLOCATE. Also, you must have had Write access to the file at the time of the ?GOPEN or ?SOPEN call.

What It Does

?ALLOCATE allocates disk blocks for specified data elements and initializes to zero those elements that do not actually exist. If the data elements already exist, the contents do not change and ?ALLOCATE takes the normal return. ?ALLOCATE pends only the task within the process that issues ?ALLOCATE.

The operating system determines the actual number of data elements to allocate from the information that u supply in the following ?RDB/?WRB packet offsets:

?PSTI	Right byte: number of data blocks to allocate Left byte: ignored
?PRNH	Starting data block number (doubleword)

?ALLOCATE Continued

The operating system ignores all other offsets in the ?RDB/?WRB packet.

Note that to use ?ALLOCATE, the file must have been opened with ?GOPEN or ?SOPEN, not with ?OPEN.

Notes

- See the descriptions of ?GOPEN, ?SOPEN, and ?RDB/?WRB in this chapter.

?ASSIGN — Assigns a character device to a process.

AOS/VS

?ASSIGN
error return
normal return

Input

AC0	Byte pointer to the name of the device to assign
AC1	Reserved (Set to 0.)
AC2	Reserved (Set to 0.)

Output

AC0	Unchanged
AC1	Undefined
AC2	Undefined

Error Codes in AC0

ERASS	Assign error — already your device
ERIDT	Illegal device type
ERIFL	IAC (Intelligent Asynchronous Controller) failure
ERVBP	Invalid byte pointer passed as a system ll argument

Why Use It?

?ASSIGN allows you to link a character device exclusively to your process until you issue ?DEASSIGN or your process terminates. You can issue ?ASSIGN against a device if you want to close it periodically without the risk of losing it to another process.

Who Can Use It?

There are no special process privileges needed to issue this call, and there are no restrictions concerning file access.

What It Does

?ASSIGN assigns the character device that you specify in AC0 to the calling process. When you assign a device to a process, you are reserving the device for the exclusive use of that process. Once you assign a device to a process, it remains the exclusive property of that process until the process terminates or breaks the assignment by issuing ?DEASSIGN. You cannot assign a device that is currently assigned to another process (such as a device enabled for spooling.)

Notes

- See the description of ?DEASSIGN in this chapter.
- See the descriptions of ?OPEN and ?CLOSE in this chapter for information on implicit device assignments and deassignments.

AOS/VS

?AWIRE
error return
normal return

Input

AC0	Not used (Set to 0.)
AC1	Set ?AWENT to wire the entire Agent; Set ?AWUDS to wire only the areas of the Agent required to support user devices
AC2	Reserved (Set to 0.)

Output

AC0	Unchanged
AC1	Unchanged
AC2	Unchanged

Error Codes in AC0

ERPTY	Illegal process type
ERPRE	Invalid parameter passed as system call argument

Why Use It?

Use the ?AWIRE system call to unwire Agent pages if your process is resident for the purpose of supporting user devices. This will free up several pages of physical memory which can be used for other purposes. While this may seem to degrade efficiency of a resident process, it actually increases the efficiency of the system as a whole.

Who Can Use It?

There are no special process privileges needed to issue this call, and there are no restrictions concerning file access.

What It Does

This call allows a resident process to change the wiring characteristics of the Agent portion of the operating system. Processes that ?PROC declares resident and processes that do a ?CTYPE against themselves to become resident will, by default, have their entire Agent wired. This is done to decrease the number of page faults taken by resident processes, thereby increasing their efficiency. However, resident processes that wish to support user–defined devices, and do not need the increased efficiency of a wired Agent can decrease their working set size by issuing the ?AWIRE system call with ?AWUDS set in AC0. This will cause the Agent to unwire all pages, except those required for user device support. This will also free up several pages of physical memory so they can be used for other purposes. If, after issuing the ?AWIRE system call with ?AWUDS set in AC0, the process would like to wire the entire Agent again to increase efficiency, the process can reissue the ?AWIRE system call with ?AWENT set in AC0. This will cause the Agent to wire its entire address space. If multiple ?AWIRE system calls are issued with the same option, they will take the normal return, but will have no effect.

?BLKIO *[packet address]*
error return
normal return

Operating System Differences

AOS/RT32 does not support modified sector I/O.

Input

AC0	Reserved (Set to 0.)
AC1	Reserved (Set to 0.)
AC2	Address of the ?BLKIO packet unless you pass the address as an argument to the system call.

Output

AC0	Undefined
AC1	Undefined
AC2	Address of the packet.

Error Codes in AC0

ER32U	Record too large for this device when using ?BM32R for a 32–bit record number
ERDIO	Attempt to issue MCA direct I/O with outstanding requests
EREOF	End of file
ERVWP	Invalid word pointer passed as system call argument
ERPUF	Physical unit failure
ERFAD	File access denied
ERRAD	Read access denied
ERSPC	File space exhausted
ERWAD	Write access denied
ERIOT	Wrong type I/O for open type
ER_FS_DIRECTORY_NOT_AVAILABLE	Directory not available because the LDU was force released (AOS/VS II only)
ER_FS_TLA_MODIFY_VIOLATION	Attempt to modify file with ?ODTL value supplied in ?GOPEN packet (write block only)

Why Use It?

Use ?BLKIO for block reads or writes, or for physical block reads or writes (see ?RDB/?WRB and ?PRDB/?PRWB.) ?BLKIO is particularly useful for its read next allocated element feature when you're reading files with many unallocated elements.

Who Can Use It?

There are no special process privileges needed to issue this call. You must have Read access to the file that you want to read from and Write access to the file that you want to write to.

What It Does

?BLKIO performs block I/O on a file. This system call combines the functionality of the ?RDB/?WRB (block I/O) and ?PRDB/?PWRB (physical block I/O) system calls, and includes some additional functionality. When you use ?BLKIO for a block read, a block write, a physical block read, or a physical block write, the system call acts EXACTLY like the ?RDB/?WRB and ?PRDB/?PWRB system calls. (For details, refer to the sections on these system calls in this chapter.)

In addition, ?BLKIO allows you to read the next allocated element in a disk file skipping unallocated blocks. Writing to any disk block in an element allocates the element to be read. Note that the read next allocated element option can only be used if the call is for a read and physical I/O has not been selected.

The ability to read the next allocated element is especially useful if you're reading a large file with many unallocated elements; ?BLKIO skips over these elements and reads only from the allocated ones. As a result, if you read a long file using this option, you save time. (?RDB reads every element, allocated or not.) You can see some illustrations of this feature later in the discussion of ?BLKIO.

Before you can issue ?BLKIO against a file, you must open the file with the ?GOPEN system call. You must also set up a parameter packet of length ?BLTH in your address space. Figure 2–1 shows the structure of the packet, and Table 2–2 describes its contents.

Under AOS/VS only, ?BLKIO also allows you to perform modified sector I/O on disks that support this hardware feature. Modified sector I/O must be used in conjunction with physical block I/O. This type of I/O allows you to read and to clear only the modified blocks of a disk unit. However, modified sector I/O does not function under AOS/VS II.

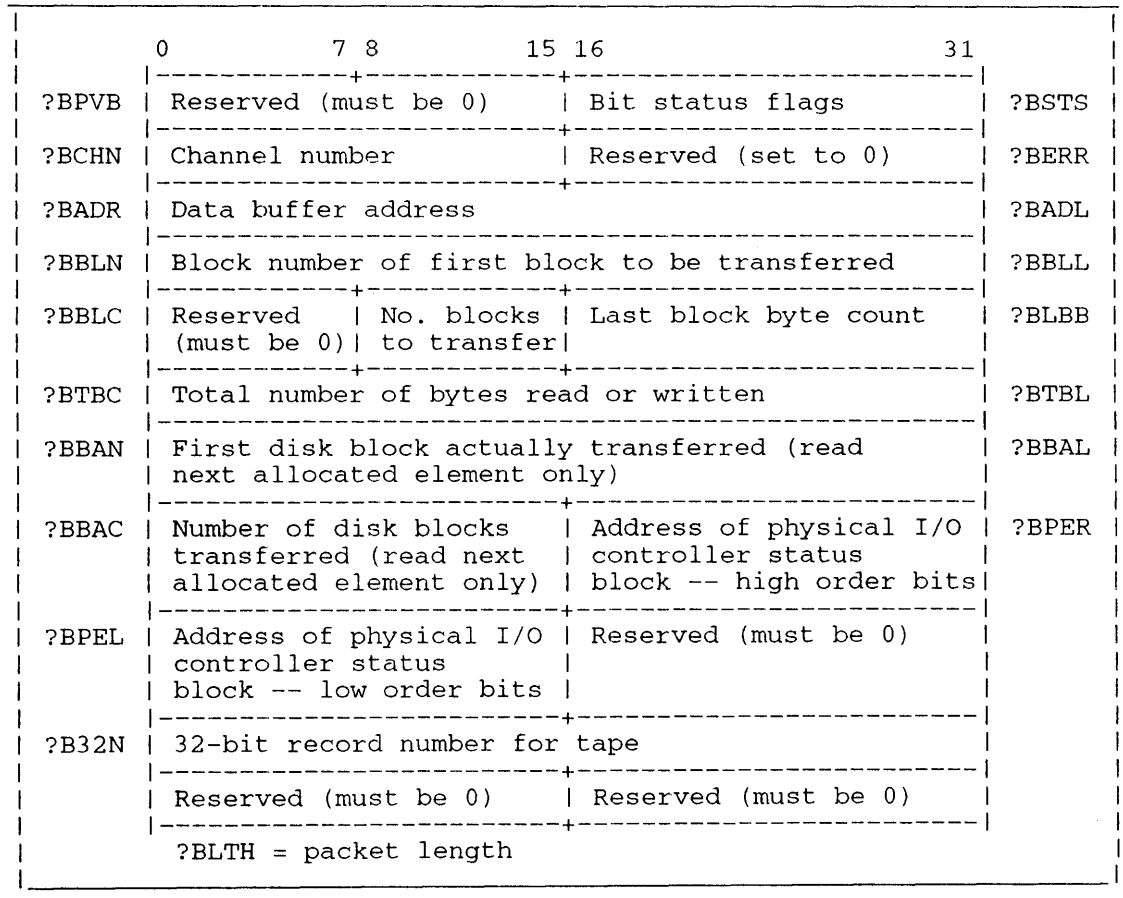


Figure 2–1. Structure of ?BLKIO Packet

Table 2–2. Contents of ?BLKIO Packet

Offset	Contents
?BPVB	Reserved (must be 0).
?BSTS	To write a block use ?BMIO To perform physical block I/O use ?BMPIO To read the next allocated element use ?BMNAB To perform modified sector I/O use ?BMNMB To perform extended I/O (32-bit record #) use ?BM32R (AOS/VS II 2.10 only.) When the bit is set to 1, the system call uses a 32-bit record number contained in ?B32N. If the bit is not set, the operating system assumes it is a 16-bit request. The operating system gets the record number from the field ?BBLN, which contains a file number and record number. If the ?B32N bit is set to 1 and the ?BLKIO system call packet is not for tape, the operating system ignores the bit. Other bit status flags are ?BMDIO = direct I/O for MCAs. ?BMEOR = enable VFU load/end of tape override. ?BMAFE = safe write block for magnetic tape.
?BCHN	Channel number (obtained with ?GOPEN).
?BERR	Reserved (must be 0).
?BADR	Address of data buffer in your address space (high-order bits).
?BADL	Address of data buffer (low-order bits).
?BBLN	File block number of first disk block to be transferred (high-order bits); file number or -1 for magnetic tape; link number for MCA.
?BBLL	File block number of first disk block to be transferred (low-order bits); block number for tape; retry count for MCA.

(continued)

Table 2–2. Contents of ?BLKIO Packet

Offset	Contents
?BBLC	High byte: reserved (must be 0). Low byte: number of blocks to transfer.
?BLBB	Number of bytes in the last disk or tape block transferred; number of bytes in last MCA transmission.
?BTBC	Total number of bytes read or written (high-order bits).
?BTBL	Total number of bytes read or written (low-order bits).
?BBAN	Block # of first disk block transferred--for read next allocated element only. (high order bits.)
?BBAL	Block # of first disk block transferred (high order bits.)
?BBAC	Number of disk blocks actually transferred--for read next allocated element only.
?BPER	Address of physical I/O controller status block (high-order bits).
?BPEL	Address of physical I/O controller status block (low-order bits).
	Reserved (must be 0).
?B32N	32-bit record number for tape. When ?BM32R is set to 1, use this offset for the magnetic tape record number.
	Reserved (must be 0).
	Reserved (must be 0).

(concluded)

Offset ?BSTS must contain a command to indicate the kind of I/O you wish to perform. For example, to write a physical block, you must use the masks ?BMIO and ?BMPIO. To read the next allocated block, just use the mask ?BMNAB.

?BSTS may also include some optional flags for magnetic tapes and MCAs. For details, see ?RDB/?WRB in this chapter.

Offset ?BADR points to the data buffer you reserved in your logical address space for the block I/O transfer. If you read or write disk blocks, use offsets ?BBLN and ?BBLL to indicate the relative block number of the first disk block you want to transfer. (For details of how ?BBLN and ?BBLL are used with tapes and MCAs, see ?RDB/?WRB in this chapter. Note that ?BBLN and ?BBLL correspond to ?PRNH and ?PRNL, respectively.)

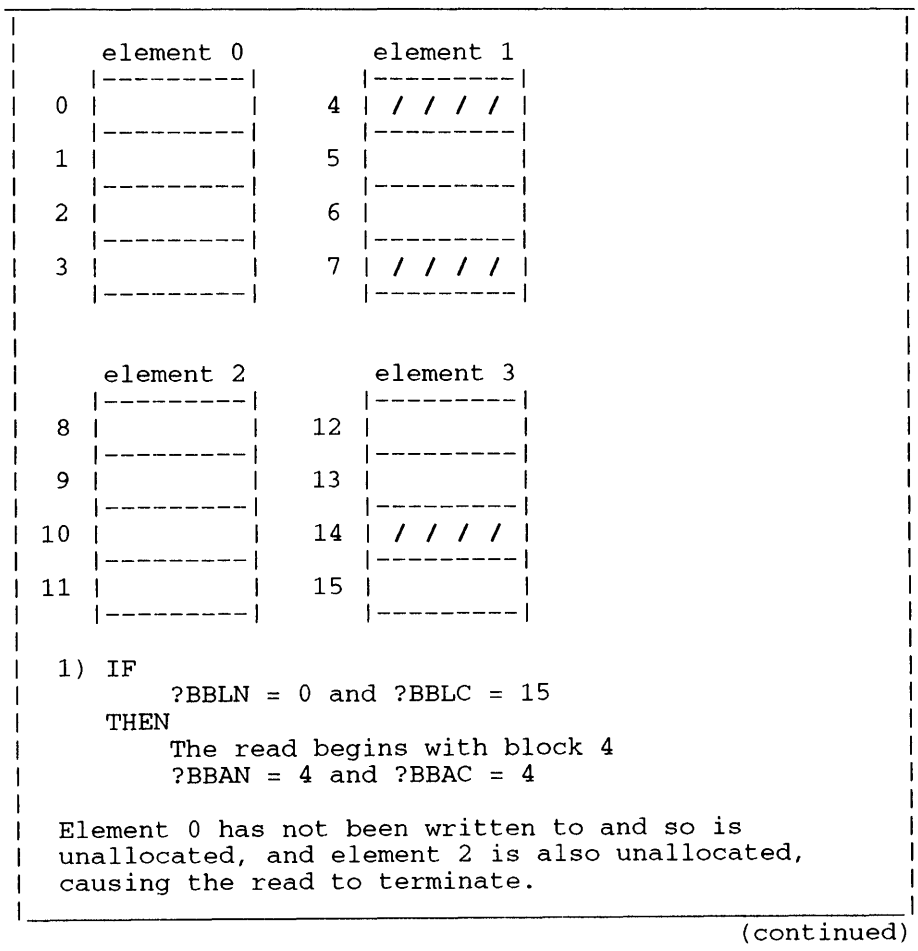
If you are writing disk blocks, use offset ?BLBB to indicate the number of bytes in the last block you wish to transfer. If you set this offset to 0, the system sets the bytes in the last block to 512, the

default. (For using ?BLBB with tapes or MCAs, see ?RDB/?WRB in this chapter. Note that ?BLBB corresponds to ?PRCL.)

Reading Allocated Blocks

If you use mask ?BMNAB in offset ?BSTS, ?BLKIO reads the next allocated element in a disk file. With this option, the call uses the starting block number that you've indicated in offset ?BBLN and the number of blocks you want transferred (?BBLC), to determine where in the file to begin the read for the next allocated element. This form of the ?BLKIO call returns the file block number of the first block in the next allocated element into offset ?BBAN and the number of blocks it actually transferred into offset ?BBAC and the last byte count in ?BLBB. See Figure 2–2 for examples. The value of ?BBAC is one of the following, depending on which happens first. Blocks will be read until an unallocated block is reached, the number of blocks specified in offset ?BBCL has been satisfied, or the end–of–file is reached, whichever occurs first.

Figure 2–2 gives some examples of how the read next allocated element option works. The illustration assumes 4–block elements (that is, an element size of 4), with slashes indicating that a block has been written to.



(continued)

Figure 2–2. Examples of Read Next Allocated Element Option

```
2) IF
        ?BBLN = 5 and ?BBLC = 15
   THEN
        The read begins with block 5, and
        ?BBAN = 5 and ?BBAC = 3

Element 2 is unallocated, so the read ends. ?BBAN =
5 because, although block 5 has not been written to,
other blocks (4 and 7) in the element have, which
means Element 1 is allocated.

3) IF
        ?BBLN = 7 and ?BBLC = 15
   THEN
        The read begins with block 7, and
        ?BBAN = 7 and ?BBAC = 1

Element 2 is unallocated, so the read ends.

4) IF
        ?BBLN = 2 and ?BBLC = 3
   THEN
        The read begins with block 4, and
        ?BBAN = 4 and ?BBAC = 3

Element 0 is unallocated, so element 1 is used. The
read terminates because the number of blocks
requested (?BBLC or 3) has been reached since it is
less than or equal to the number of blocks (4) in
the element.

5) IF
        ?BBLN = 10 and ?BBLC = 30
   THEN
        The read begins with block 12
        ?BBAN = 12 and ?BBAC = 4

Element 2 is unallocated, so Element 3 is used, but
the file ends before the read can be satisfied, so
EREOF (end of file) is returned.
```

(concluded)

Figure 2–2. Examples of Read Next Allocated Element Option

Physical Block I/O

For ?BLKIO to perform physical block I/O on a disk or tape unit, you must select ?BMPIO in offset ?BSTS, plus indicate whether you want to write or read (?BMIO to write). In addition, offset ?BPER must point to a block of words that you have reserved for controller status information about the block transfer. The block must be ?BPBLT words long. Figure 2–3 shows the structure of the block.

Into this reserved block, ?BLKIO returns the relative block number of the last block transferred, plus eight controller status words. The words ?BRBB through ?BCS8 are identical to offsets ?PRBB through ?PCS8 in the ?PRDB/?PWRB packet (see Figure 2–3 and Table 2–157 for details). For interpretations of this status block, refer to Table 2–158 and Table 2–159 in the description of system calls ?PRDB and ?PWRB.

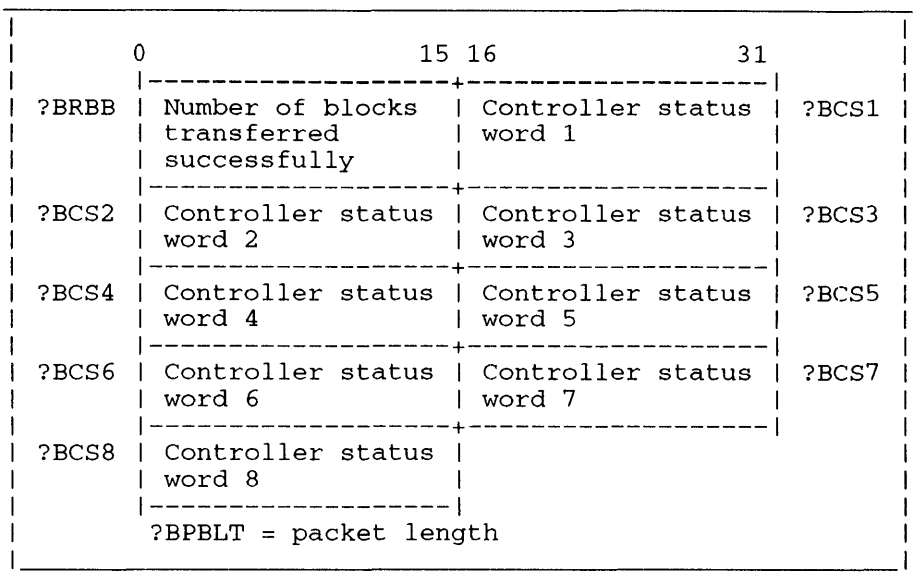


Figure 2–3. Structure of Physical I/O Controller Status Block

For ?BLKIO to perform modified sector I/O to a disk unit, you must select ?BMNMB in offset ?BSTS. Since modified sector I/O works in conjunction with physical block I/O, ?BMPIO in offset ?BSTS must also be selected (refer to physical block I/O discussion, above).

When performing modified sector I/O, offset ?BBLC must contain the number of blocks to transfer, plus one block for the modified sector bit map. The first block (256 words) of the transfer contains the modified sector bit map. For more information on modified sector I/O, refer to the manual *Programmer's Reference Series: Models 6236/6237 and Models 6239/6240 Disk Subsystems.*

NOTE: Modified sector I/O is only valid when used on a disk that supports this hardware feature. Currently, this support is only available for 354–megabyte and 592–megabyte "DPJ" disk subsystems. However, modified sector I/O does not function under AOS/VS II.

Notes

- See ?RDB/?WRB and ?PRDB/?PWRB in this chapter.

?BLKPR
error return
normal return

Input

AC0	One of the following: • Byte pointer to the name of the target process • PID of the target process • –1 to block the calling process (This causes the OS to ignore AC1.)
AC1	One of the following: • –1 if AC0 contains a byte pointer • 0 if AC0 contains a PID • AC1 is ignored if AC0 contains –1
AC2	Reserved (Set to 0.)

Output

AC0	Unchanged
AC1	Unchanged
AC2	Undefined

Error Codes in AC0

ERMPR	System call parameter address error
ERPRE	Invalid system call parameter, AC0 was not –1 and AC1 was not 0 or –1
ERPNM	Illegal process name format
ERPOR	PID is out of range for this process
ERPRH	Attempt to access a process not in hierarchy
ERPRV	You must have Superprocess mode turned on to access another process
ERVBP	Invalid byte pointer to name of target process passed in AC0

Why Use It?

?BLKPR gives you partial control over the system scheduler. You can use ?BLKPR to block one process in favor of another process, or to stop a large job temporarily.

Who Can Use It?

A process can issue ?BLKPR to block any process subordinate to it. However, if the calling process is in Superprocess mode, it can block any process. There are no restrictions concerning file access.

What It Does

?BLKPR allows the calling process to block itself.

When the operating system blocks a process, it also suspends all tasks in the process until a ?UBLPR system call is issued against that process. Suspension on that event is independent of — and has no effect on — the state of any other suspensions that may be in force against a task.

Notes

- See the description of ?UBLPR in this chapter.

?BNAME — Determines whether process name/queue name is on local or remote host.

?BNAME
error return
normal return

Input

AC0	Byte pointer to the process name/queue name
AC1	Reserved (Set to 0.)
AC2	Reserved (Set to 0.)

Output

AC0	One of the following: • 0 if process name/queue name is on local host • Bits 17 through 31 contain host ID if process name/queue name is on remote host • –1 if reference is remote and host has zero host ID
AC1	Unchanged
AC2	Unchanged

Error Codes in AC0

ERPNM Illegal process name
The input process name/queue name does not contain a colon (:).

Why Use It?

You can use ?BNAME to find out if a particular process or queue is on a local host or on a remote host.

Who Can Use It?

There are no special process privileges needed to issue this call, and there are no restrictions concerning file access.

What It Does

?BNAME determines whether a particular process or queue is on a local or a remote host. To use ?BNAME, you must specify a process name or queue name in AC0. Then, the operating system examines the name to decide whether the process/queue is on a local or a remote host. If the host is remote, or if the process name or the queue name contains the local host name, the operating system returns the host ID in Bits 17 through 31 of AC0.

If the host ID does not exist in directory :NET, ?BNAME assumes the case of a local host.

?BRKFL Terminates a process and creates a break file.

AOS/VS

?BRKFL
error return
normal return

Input

AC0 One of the following:

- Byte pointer to the name of the target process
- PID of the target process
- –1 to terminate the calling process

AC1 One of the following:

- –1 if AC0 contains a byte pointer
- 0 if AC0 contains a PID
- AC1 is ignored if AC0 contains –1

AC2 One of the following:

- Byte pointer to the break file's filename
- –1 to use the default break file filename

Output

AC0 Unchanged

AC1 Unchanged

AC2 Unchanged

Error Codes in AC0

ERPRH	Attempt to access a process not in hierarchy
ERVBP	Invalid byte pointer to name of target process passed in AC0

Why Use It?

?BRKFL is useful when you are debugging a program, because it allows you to save the current state variables of a terminated process.

Who Can Use It?

The calling process must have Write or Append access to the target process's initial directory. The process that is terminating must have Write or Append access to its initial directory. The calling process can request a break file only for itself or for a subordinate process. In Superprocess mode, the calling process can request a break file for any process.

?BRKFL Continued

What It Does

?BRKFL terminates the process that you specify in AC0 (the caller or another process) and creates a break file in the terminated process's initial directory. The break file contains the current state of the target process at the time that it terminates. Specifically, the operating system copies the following words to the break file:

Status Word	Contents
BRAC0	Value of AC0
BRAC1	Value of AC1
BRAC2	Value of AC2
BRAC3	Value of AC3
BRPC	Value of the program counter (PC)
BRTID	Task identifier (TID)
BRFP	Value of the stack frame pointer
BRSP	Value of the stack pointer
BRSL	Value of the stack limit
BRSB	Value of the stack base

(Refer to the current AOS/VS Release Notice for more information on the contents of a break file.)

When you issue ?BRKFL, either load AC2 with a byte pointer to the filename you want to assign to the break file, or use the system–generated default filename. The default filename is

?pid.time.BRK

where

pid — is the 5–digit PID of the target process.

time — is the time of the termination (and break file creation) in the form hours_minutes_seconds.

If the operating system cannot create the break file (for example, if creating a break file causes the target process's initial directory to exceed its disk space, or if the caller lacks proper access to the target process's initial directory), it takes the error return for ?BRKFL. Note, however, that the target process still terminates.

You can terminate your current process and create a break file by typing Ctrl–C Ctrl–E at your terminal. Also, you can direct the operating system to create a break file when the process traps by setting the ?PBRK parameter in the ?PROC packet for a process.

Notes

- See the descriptions of ?PROC and ?MDUMP in this chapter.

?CDAY
error return
normal return

Input

AC0	Scalar date you wish to convert (from a base value of 31 December 1967) in Bits 16 through 31
AC1	Reserved (Set to 0.)
AC2	Reserved (Set to 0.)

Output

AC0	Day from 0 through 31
AC1	Month from 1 through 12
AC2	Year minus 1900 (The result is expressed in octal.)

Error Codes in AC0

No error codes are currently defined.

Why Use It?

Although the system calendar maintains the current date as day, month, and year, the date is sometimes expressed as a scalar value. For example, scalar notation appears in the ?FSTAT packet and in the time block for ?CREATE. Thus, ?CDAY can be a useful conversion tool for both ?FSTAT and ?CREATE.

Who Can Use It?

There are no special process privileges needed to issue this call, and there are no restrictions concerning file access.

What It Does

?CDAY converts the scalar value for a date that you specify in AC0 to its equivalent in days, months, and years. The scalar value for a date equals the number of days that have elapsed since 31 December 1967 (the base value).

?CGNAM
error return
normal return

Input

AC0	Byte pointer to a receive buffer for the pathname
AC1	Channel number of the target file
AC2	Byte length of the receive buffer

Output

AC0	Unchanged
AC1	Unchanged
AC2	Actual length of the pathname, excluding the terminator

Error Codes in AC0

ERCIU	Channel in use
ERFDE	File does not exist
ERFNO	Channel not open
ERICN	Illegal channel number
ERIRB	Insufficient room in buffer (The buffer that you specified in AC0 is too small to accommodate the pathname.)
ERMPR	System call parameter address error
ERVBP	Invalid byte pointer passed as a system call argument
ER_FS_INVALID_PATHNAME_BYTE_PTR	Invalid byte pointer to pathname

Why Use It?

A file's pathname identifies its position in the directory structure. Therefore, you need the pathname as input for many of the file–maintenance system calls. ?CGNAM is a simple tool that allows you to get this information.

Who Can Use It?

There are no special process privileges needed to issue this call. You must have obtained a channel number, via ?OPEN or ?GOPEN, before issuing ?GGNAM. There are no other restrictions concerning file access.

What It Does

?CGNAM returns the complete pathname of the file whose channel number you specify in AC1. Before you issue ?CGNAM, set up a receive buffer for the pathname and use a byte pointer in AC0 to address it. In addition, load AC2 with the buffer length, and load AC1 with the target file's channel number. (The operating system returns the target file's channel number in the ?OPEN packet.)The operating system returns the file's complete pathname, starting with the root, to the receive buffer that you specify.

Notes

- See the description of ?OPEN in this chapter.

?CHAIN Passes control from a Ring 7 caller to a new program.

?CHAIN
error return

Input

AC0	Byte pointer to the new program's pathname
AC1	Flag bits (all others = 0): Bit 30 = 1 to flush outstanding IPC messages Bit 30 = 0 to retain (do not flush) outstanding IPC messages Bit 31 = 0 to pass control to the new program's entry point Bit 31 = 1 to pass control to the user debugger
AC2	One of the following: • 0 if there is no IPC message • Address of the IPC message header for the new program, if there is an IPC message

Output

AC0	Undefined
AC1	Undefined
AC2	Undefined

Error Codes in AC0

EREAD	Execute access defined
ERICH	B– or C–type process cannot chain to the target program
ERMEM	Insufficient memory available
ERVBP	Invalid byte pointer to name of rget process passed in AC0
ERVWP	Invalid word pointer as a system call argument

Why Use It?

?CHAIN is a way of tying together several steps of a long, complex program set, where each program is a separate segment image. ?CHAIN's main function is to provide compatibility with 16–bit AOS programs, as the 32–bit address space of the MV Family provides room for almost unlimited code and data. In general, try to minimize the use of ?CHAIN, especially in new applications.

In a customer/server relationship, you can use ?CHAIN to tie a customer process to a new program. As long as the connection is from Ring 7, ?CHAIN does not affect the customer process's connection with the server. The operating system sends the server an IPC message to notify it when a customer process issues ?CHAIN. Also, ?CHAIN preserves connections for Ring 3 when a customer chains (provided the connection existed before the ?CHAIN).

?CHAIN Continued

Do not issue ?CHAIN while a customer process has requests to servers outstanding. Also, if a customer process chains to a new program, do not issue ?MBFC or ?MBTC against that customer process, unless you can verify the validity of the customer's buffer.

Who Can Use It?

There are no special process privileges needed to issue this call. You must have Execute access to the new program's file.

What It Does

?CHAIN must originate from a Ring 7 segment image only. When a Ring 7 process issues ?CHAIN, the operating system releases the resources that the process was using, and then executes a new program. ?CHAIN transfers the following attributes to the new program:

- The username, process name, PID, terminal, search list, and working directory of the calling process.
- The generic file associations of the calling process (for example, the filenames associated with the generic files @INPUT, @OUTPUT, @LIST, and @DATA).
- The privileges, process type, and priority of the calling process in Ring 7.

?CHAIN does not pass open files, protected shared file grants, or inner rings to the new program. Instead, it closes all files and flushes all unfulfilled I/O. It also removes any user–defined device definitions and destroys inner ring segments. As a result, the ?CHAIN has the effect of terminating all processes created by the inner rings and breaking all connections involving these inner rings.

You can ?CHAIN between processes of different types (i.e., 16–bit processes and 32–bit processes). This is useful if you lack the privilege to create unlimited sons.

Do not set Bit 30 of AC1 if you want the operating system to pass outstanding IPC messages to the new program. If you want to send a specific IPC message, load AC2 with the address of the IPC header.

Set Bit 31 of AC1 if you want the new program to start in the debugger. Note, however, that if the calling program was started in the debugger, then the new program always starts in the debugger, regardless of the value of Bit 31.

The contents of each accumulator is undefined after a ?CHAIN. Because control passes to the new program, ?CHAIN has no normal return.

?CKVOL Checks volume identifier of a labeled magnetic tape.

AOS/VS

?CKVOL
error return
normal return

Input

AC0	Byte pointer to the unit's pathname	
AC1	Byte pointer to the volume identifier (volid) string	
AC2	Bit masks:	
	?XMIBM	if the labels are in IBM format
	?OPDL	if the density mode is 800 bpi
	?OPDM	if the density mode is 1600 bpi
	?OPDH	if the density mode is 6250 bpi
	?OPAM	for automatic density mode selection

Output

AC0	Unchanged
AC1	Unchanged
AC2	Unchanged

Error Codes in AC0

ERVOL Incorrect labeled tape volume mounted

File system error codes

Why Use It?

?CKVOL allows you to make sure the correct labeled tape volume is mounted before you read from or write to the tape.

Who Can Use It?

There are no special process privileges needed to issue this call, and there are no restrictions concerning file access.

What It Does

?CKVOL checks the volume identifier (volid) of a labeled magnetic tape against a volid that you (the user) supply.

?CKVOL compares the volid that AC1 points to with the volid of the labeled tape that is mounteon the unit whose pathname AC0 points to. If these two volids are not the same, the operating system returns error code ERVOL in AC0. If the volids are the same, the operating system takes the normal return.

AOS/VS

?CLASS *[packet address]*
error return
normal return

Input

AC0	Reserved (Set to 0.)
AC1	Reserved (Set to 0.)
AC2	Address of the ?CLASS packet, unless you specify the address as an argument to ?CLASS

Output

AC0	Unchanged
AC1	Unchanged
AC2	Address of the ?CLASS packet

Error Codes in AC0

ERCL0	Cannot delete class 0
ERCLU	Class in use
ERIMP	Illegal bit map
ERMPR	Parameter address error
ERPRV	Caller not privileged for this action

Why Use It?

Use this system call to receive or supply the class IDs that AOS/VS uses to schedule classes.

Who Can Use It?

For the "get" version of this call, there are no special process privileges needed to issue it, and there are no restrictions concerning file access. For the "set" version of this call, you must have System Manager privilege to issue it, but there are no restrictions concerning file access.

What It Does

This system call's parameter packet contains get/set code and a class bit map. The get/set code indicates whether AOS/VS will obtain the class bit map or else will change the class bit map and also change the current configuration of classes. The class bit map identifies which classes are legal; you may then specify them in the class scheduling matrix and LP class hierarchy. There are 16 classes with IDs between 0 and 15, inclusive.

Suppose, in response to a "get" code, AOS/VS sets a bit in the class bit map. This means the corresponding class is valid. For example, if AOS/VS has set bits 0, 4, and 5 in the map, then classes with ID values 0, 4, and 5 are valid. Other classes, such as those with ID values 1 and 3, are invalid and will cause errors if you try to use them.

If you want to change the class IDs that AOS/VS is currently using, issue the ?CLASS call with a "set" code. AOS/VS will read the class bit map. It will add/delete classes according to the bits you set. However, you cannot delete a class that appears in the class scheduling matrix; neither can you delete class 0 — the class that exists when AOS/VS first comes up.

Figure 2–4 shows the structure of the ?CLASS parameter packet, and Table 2–3 describes its contents.

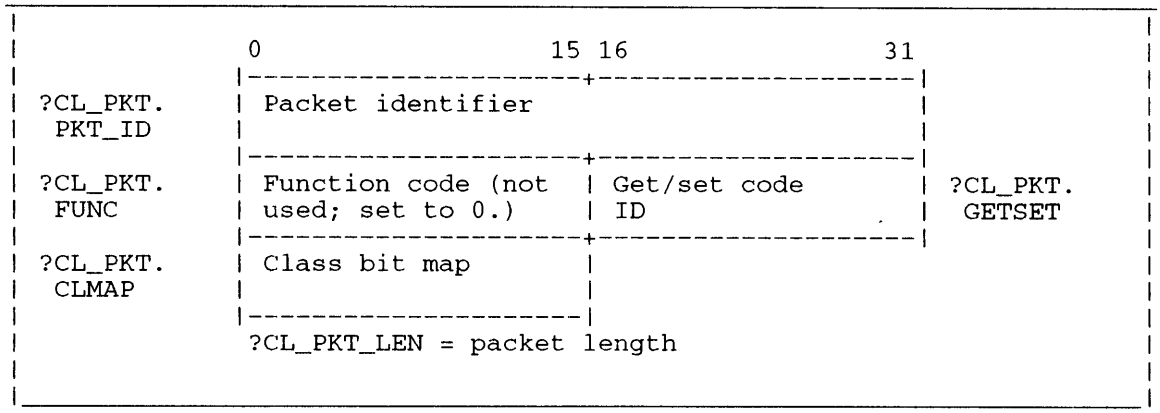


Figure 2–4. Structure of ?CLASS Packet

Table 2–3. Contents of ?CLASS Packet

Offset	Contents
?CL_PKT.PKT_ID (doubleword)	Packet identifier. Place ?CL_PKT_PKTID here.
?CL_PKT.FUNC	Function code. Not used (set to 0).
?CL_PKT.GETSET	Get/set code. Place the value of ?CL_GET here to obtain the class bit map. Place the value of ?CL_SET here to update the bit map; also place the new bit map in offset ?CL_PKT.CLMAP.
?CL_PKT.CLMAP	The class bit map. You obtain it or set it according to the value in offset ?CL_PKT.GETSET.

?CLOSE — Closes an open channel.

?CLOSE *[packet address]*
error return
normal return

Input

AC0	Reserved (Set to 0.)
AC1	Reserved (Set to 0.)
AC2	Address of the ?CLOSE packet, unless you specify the address as an argument to ?CLOSE

Output

AC0	Undefined
AC1	Undefined
AC2	Address of the ?CLOSE packet

Error Codes in AC0

ERACU	Attempt to close unopen channel/device
ERCIU	Channel in use
ERPRE	Invalid system call parameter
ERSIM	Simultaneous requests on same channel
ERVWP	Invalid word pointer passed as a system call argument

Why Use It?

After you read from or write to an open I/O channel, issue ?CLOSE to close the channel. This allows the operating system to update the associated file's status information and to free the channel so that it can be used again.

Normally, if a process terminates, the operating system automatically closes all its open channels. However, if a process terminates abnormally (for example, because of a Ctrl–C Ctrl–B sequence or a user trap), the operating system flushes all buffers of any open channels before it closes the channels.

When you issue ?CLOSE against a labeled tape file, the operating system writes the user trailer labels from the area that you specify in offset ?ELUT of the ?OPEN packet extension.

Who Can Use It?

There are no special process privileges needed to issue this call. You must have obtained a channel number, via ?OPEN or ?GOPEN, before issuing ?CLOSE. There are no other restrictions concerning file access.

What It Does

?CLOSE closes an open channel and frees the channel number that the operating system assigned to that channel.

?CLOSE requires the I/O packet shown in Figure 2–5. You can specify the address of the packet as an argument to ?CLOSE, or you can load AC2 with a byte pointer to the packet address before you issue ?CLOSE.

As Figure 2–5 shows, ?CLOSE uses the same I/O packet as ?OPEN, ?READ, and ?WRITE. Only one I/O parameter applies to ?CLOSE:

- Channel number, offset ?ICH, which is the number of the channel you want to close. (The operating system returns the channel number to offset ?ICH after the ?OPEN.)

Also, there are no extended packets for ?CLOSE.

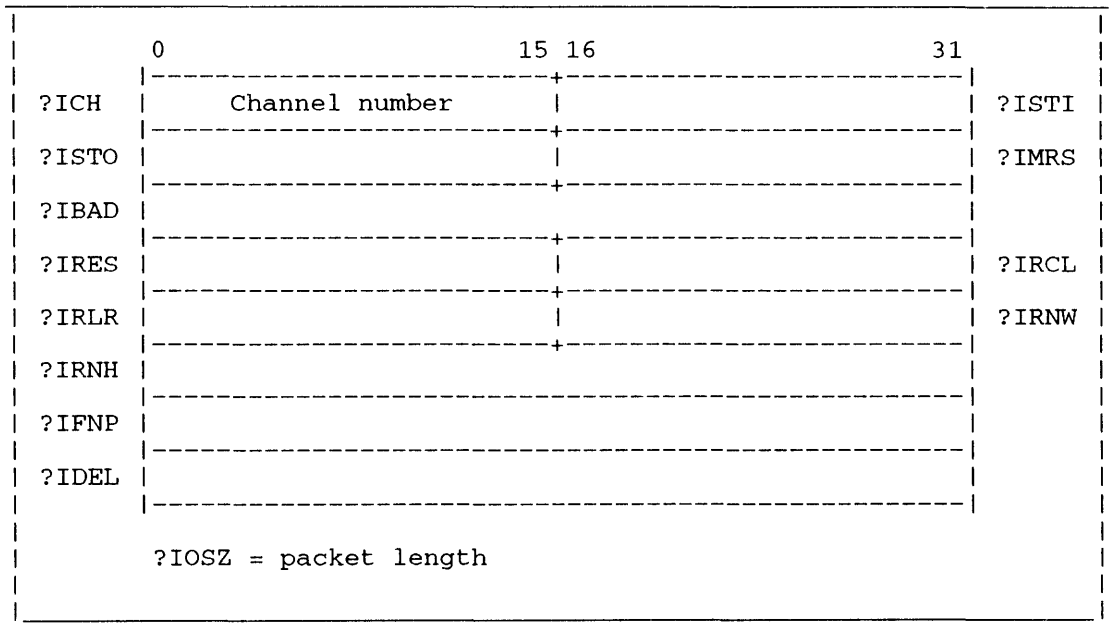


Figure 2–5. Structure of ?CLOSE Packet

Notes

- See the descriptions of ?OPEN, ?READ, and ?WRITE in this chapter for more information on I/O parameters.

?CLRDV
error return
normal return

Input

AC0 contains either:

- Byte pointer to the terminal name
- Channel (number) on which the device is opened

AC1 contains the following:

Bit 0 = 0 if AC0 contains a byte pointer

Bit 0 = 1 if AC0 contains a channel number

AC2 option:

- ?CDRXON — receive XON (simulate Ctrl–Q)
- ?CDSBRK — send a break

Output

AC0 Unchanged

AC1 Unchanged

AC2 Unchanged

Error Codes in AC0

ERFDE	File does not exist
ERICN	Illegal channel
ERIFD	Illegal function for device (Returned when trying to transmit a break to a device that does not support the transmit break function, or receiving XON on an input–only device.)
ERIFT	Illegal file type
ERPRE	Invalid system call parameter
ERPRV	Caller not privileged for this action
ERVBP	Invalid byte pointer passed as system call argument

Why Use It?

The receive XON option is useful in restarting output on a line that cannot be restarted by any of the typical methods (i.e., Ctrl–Q, hold, etc.). For example, if a line is in binary mode, it will not recognize control sequences, thus Ctrl–Q will not be interpreted as an XON command. It also is useful in restarting output on asynchronous lines that have nonkeyboard devices, such as printers. These lines may be held on Ctrl–S from characters generated by line noise.

The send break option offers a simple method of sending a break. Some devices, such as COMSWITCH, respond to a break.

Who Can Use It?

See the following section "What It Does" for an explanation of PID and target device ownership requirements.

What It Does

?CLRDV allows you two options: to simulate an XON or to transmit a break to an asynchronous line. The first option, ?CDRXON, lets you restart output by simulating and sending XON to a device. (When you type Ctrl–Q, you simulate this function.)

?CLRDV allows a process to simulate XON as long as the process is PID 2, has the System Manager privilege ON, or owns the target device.

The second option, ?CDSBRK, lets you send a break from the asynchronous controller (IAC, MCP1, etc.) to the device. The break condition lasts until a ?WRITE is issued on the line or the characteristics change. Examples of events that result in a characteristics change are a ?SECHR system call, a CLI CHARACTERISTICS command or environment POP command, and a process termination.

?CLRDV allows a process to send a break as long as the process is PID 2, has the System Manager privilege ON, or owns the target device.

The following devices support the ?CDSBRK option:

- IAC
- MCP1
- LAC
- DRT on the following machines:
  - ECLIPSE MV/1400™ DC
  - ECLIPSE MV/2000™ DC
  - ECLIPSE MV/2500™ DC

The following devices do not support the ?CDSBRK option:

- CPI/24
- The system console
- DRT on the following machines:
  - DS/4000 series
  - ECLIPSE MV/4000™ DC
  - ECLIPSE MV/4000 SC
  - ECLIPSE MV/7800™ C
  - ECLIPSE MV/7800 DCX

?CLSCHED Enables, disables, or examines class scheduling.

AOS/VS

?CLSCHED *[packet address]*
error return
normal return

Input

AC0	Reserved (Set to 0.)
AC1	Reserved (Set to 0.)
AC2	Address of the ?CLSCHED packet, unless you specify the address as an argument to ?CLSCHED

Output

AC0	Unchanged
AC1	Unchanged
AC2	Address of the ?CLSCHED packet

Error Codes in AC0

ERPRV	Caller not privileged for this action
ERCSO	Class scheduling is enabled

Why Use It?

Use this system call to enable or disable class scheduling for a specified set of LPs. You may also accumulate class scheduling statistics.

Who Can Use It?

For the enable or disable version of this call, you must have System Manager privilege to issue it, but there are no restrictions concerning file access. For the examine version of this call, there are no special privileges needed to issue it, and there are no restrictions concerng file access.

What It Does

You implement this system call in one of two basic ways: enable class scheduling with accumulation of statistics, or simply accumulate statistics about class scheduling with class scheduling disabled. You cannot enable class scheduling and disable accumulation of statistics.

For class scheduling, use this call to turn class scheduling on or off according to logical processors you specify. When an LP's class scheduling is on, AOS/VS uses the LP's class hierarchy and time interval allotments to select processes. When an LP's class scheduling is off, AOS/VS ignores the LP's class hierarchy and time interval allotments; AOS/VS also eliminates certain scheduling and accounting overhead.

For statistical accumulation, use this call to accumulate statistics that show how AOS/VS is allotting processor time. This accumulation occurs according to logical processors you specify. It always occurs when class scheduling is enabled and can occur without class scheduling if desired. Use the ?CLSTAT call to view the accumulated statistics.

Use the ?CLSCHED packet entries as follows:

- Use the get/set offset (?CLS_PKT.GETSET) to indicate whether you want to obtain the system class schedule settings or to set them. Place ?CLS_GET in the offset in the former case or ?CLS_SET in the offset in the latter case.

- Use the accumulate/schedule offset (?CLS_PKT.FLAVOR) to indicate which LP setting you wish to get/set: statistical accumulation or class scheduling.

- Offset ?CLS_PKT.LPMAP contains an LP bit map. Bit 0 corresponds to LPID 0, etc. When using the "set" code, a 1 bit tells AOS/VS to turn class scheduling or accumulation on for the corresponding LP; a 0 bit tells AOS/VS to turn class scheduling or accumulation off for that LP. AOS/VS does *not* return an error if an LP is already in the desired scheduling or accumulation mode.

  Similay, if you enter the "get" code, AOS/VS uses the LP bit map to tell you which LPs have class scheduling or accumulation enabled.

Note the following rules that apply to enabling/disabling class scheduling and statistical accumulation:

- Enabling class scheduling for an LP has AOS/VS automatically turn on statistical accumulation for the LP.

- Disabling class scheduling for an LP has AOS/VS automatically turn off statistical accumulation for the LP.

- However, you may turn on statistical accumulation for an LP, regardless of whether you have enabled or disabled class scheduling.

- You *cannot* disable statistical accumulation if the LP has class scheduling enabled.

Given these rules, the easiest way to configure your LP scheduling and accumulation is to

- Issue the scheduling version of ?CLSCHED first.

- Then issue the accumulation version of ?CLSCHED.

Figure 2–6 shows the structure of the ?CLSCHED parameter packet, and Table 2–4 describes its contents.

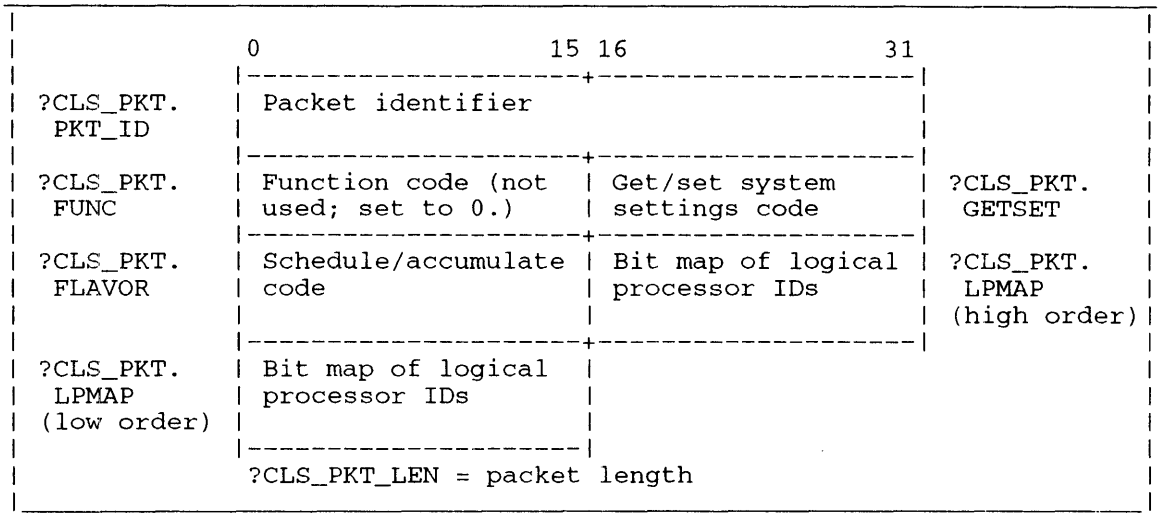


Figure 2–6. Structure of ?CLSCHED Packet

Table 2–4. Contents of ?CLSCHED Packet

Offset	Contents
?CLS_PKT.PKT_ID (double-word)	Packet identifier. Place ?CLS_PKT_PKTID here.
?CLS_PKT.FUNC	Function code. Not used (set to 0).
?CLS_PKT.GETSET	Code word into which you place ?CLS_GET when you want to obtain system settings and ?CLS_SET when you want to set system settings.
?CLS_PKT.FLAVOR	Code word into which you place ?CLS_ACC when you want to get/set LP statistical accumulation or ?CLS_SCHED when you want to get/set LP class scheduling. You can specify both values when you want to find out which logical processors are accumulating statistics.
?CLS_PKT.LPMAP (double-word)	Bit map of the logical processors, with bit 0 corresponding to LP 0, etc.

Notes

- When you initially create an LP, class scheduling and statistics accumulation are disabled.
- AOS/VS maintains scheduling statistics from the time that it creates an LP. If you turn statistics accumulation off and on, the statistics will not include information about the processes that were scheduled when the accumulation was off.
- AOS/VS returns an error if you do not have the appropriate privilege (System Manager) to issue this call.
- AOS/VS continues to maintain LP class hierarchies, class IDs, and the class matrix whether or not class scheduling is enabled. This means ?LPCLASS, ?CLASS, and ?CMATRIX calls work correctly whether or not class scheduling is in use.

AOS/VS

?CLSTAT *[packet address]*
error return
normal return

Input

AC0	Reserved (Set to 0.)
AC1	Reserved (Set to 0.)
AC2	Address of the ?CLSTAT packet, unless you specify the address as an argument to ?CLSTAT

Output

AC0	Unchanged
AC1	Unchanged
AC2	Address of the ?CLSTAT packet

Error Codes in AC0

ERICD	Illegal function code
ERLNE	LP does not exist
ERPRV	Caller not privileged for this action

Why Use It?

Use this system call to obtain information about the allocation of a logical processor's CPU time.

Who Can Use It?

There are no special process privileges needed to issue this call, and there are no restrictions concerning file access.

What It Does

The ?CLSTAT call supports a main packet and an optional subpacket. The main packet provides information about the division of an LP's CPU time among the user community. The subpacket indicates how CPU time is used among various portions of AOS/VS. You tell AOS/VS whether or not to place information in the subpacket by the value you place in offset ?CLST_PKT.FUNC of the main packet.

?CLSTAT Continued

Many of the offsets are 4 words long and contain a time value. Here is the format of the time value:

- It's 64 bits long.
- Its frequency is (2**18)*(10**4) = 2,621,440,000 cycles per second.
- Bit number n (n is between 0 and 63) changes every (2**(63–n))/((2**18)*(10**4)) seconds. So, for example:
  - Bit 45 set and the others off represents 100 microseconds.
  - Bit 31 set and the others off represents 1.6384 seconds.
  - Bit 20 set and the others off represents 3,355 seconds.
  - Bit 6 set and the others off represents about 1 year, 9 months.
- It rolls over every 223 years.

Figure 2–7 shows the structure of the ?CLSTAT main packet, and Table 2–5 describes its contents. Figure 2–8 shows the structure of the ?CLSTAT subpacket, and Table 2–6 describes its contents.

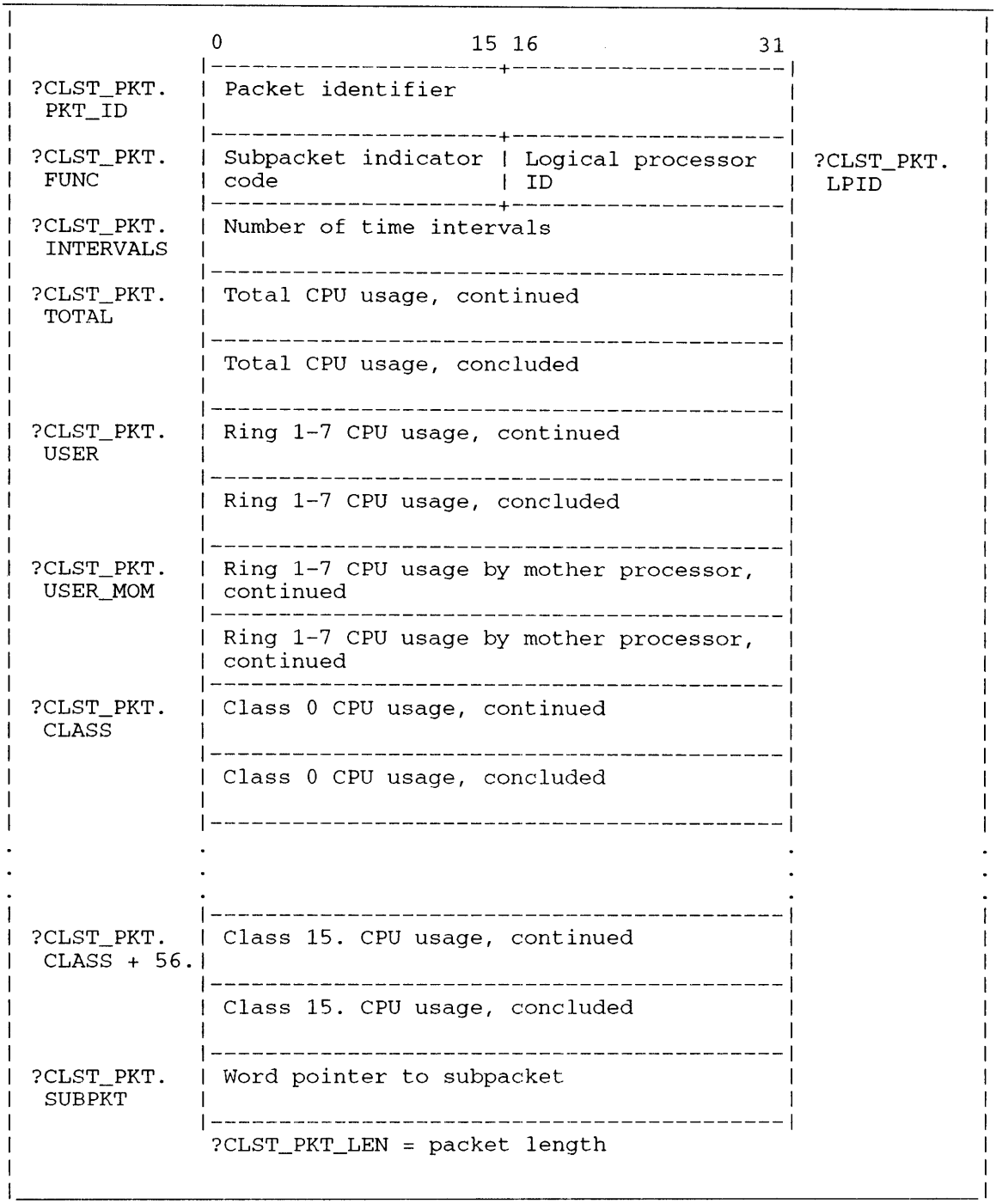


Figure 2–7. Structure of ?CLSTAT Main Packet

Table 2–5. Contents of ?CLSTAT Main Packet

Offset	Contents
?CLST_PKT.PKT_ID (double-word)	Packet identifier. Place ?CLST_PKT_PKTID here.
?CLST_PKT.FUNC	Subpacket indicator code. Place ?CLST_AOSVS here to indicate the presence of a subpacket or ?CLST_NONE to indicate its absence.
?CLST_PKT.LPID	Logical processor ID. Place the ID of the LP, about which you want class scheduling statistics, here.
?CLST_PKT.INTERVALS (doubleword)	AOS/VS returns the number of scheduling time intervals that have elapsed.
?CLST_PKT.TOTAL (four words)	AOS/VS returns the total amount of CPU time that the logical processor (i.e., its job processors) has used.
?CLST_PKT.USER (four words)	AOS/VS returns the total amount of CPU time that the logical processor has used in rings 1 through 7.
?CLST_PKT.USER_MOM (four words)	AOS/VS returns the total amount of CPU time that the logical processor has used in rings 1 through 7 on the mother processor (i.e., the LP couldn't run on a child processor). This value is included in offset ?CLST_PKT.USER.
?CLST_PKT.CLASS (four words)	AOS/VS returns the total amount of CPU time that scheduling class number 0 used.
. . .	. . .
?CLST_PKT.CLASS+56. (four words)	AOS/VS returns the total amount of CPU time that scheduling class number 15. has used.
?CLST_PKT.SUBPKT (doubleword)	If you specified a subpacket in offset ?CLST_PKT.FUNC, then place its word address here.

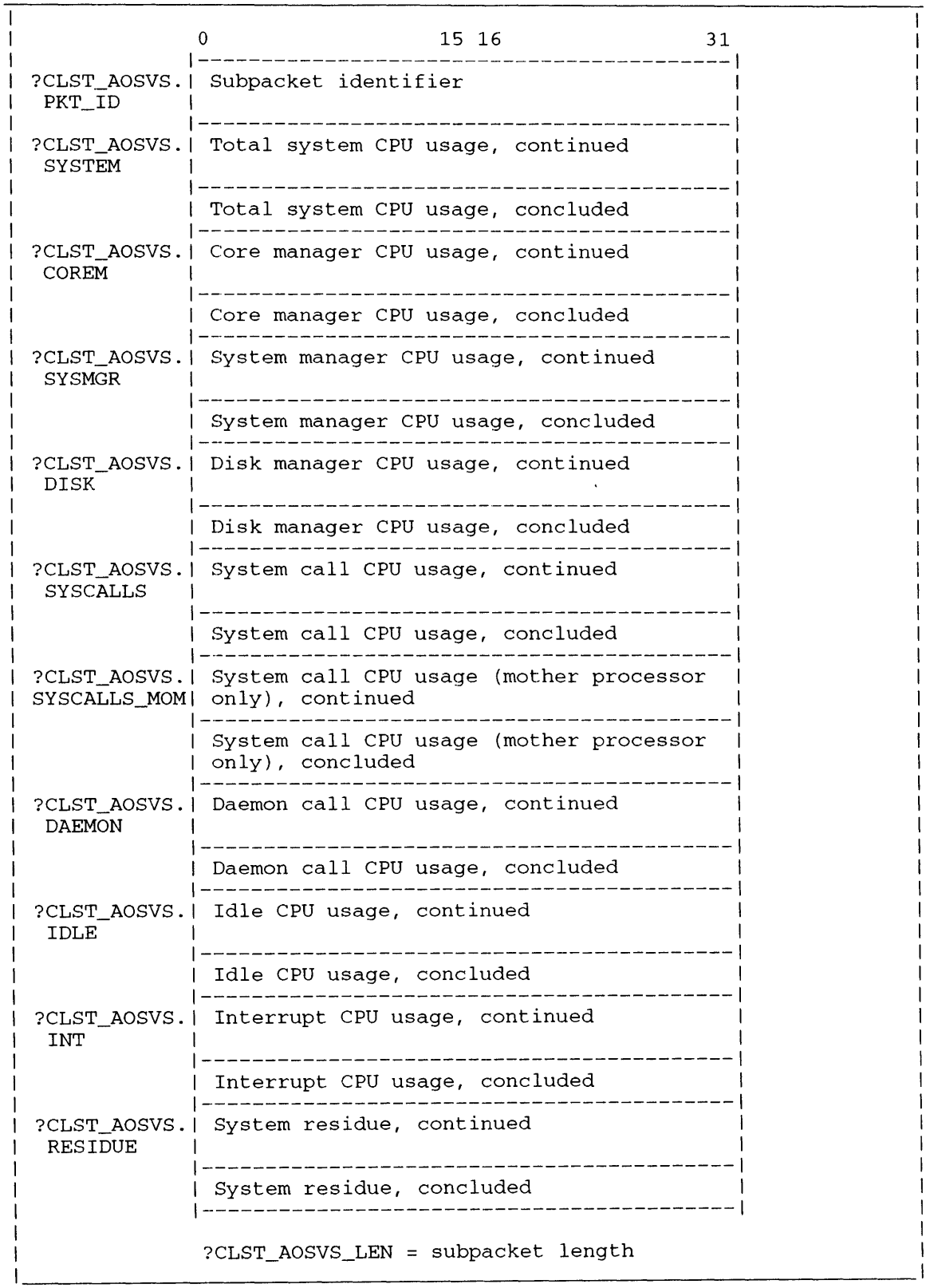


Figure 2–8. Structure of ?CLSTAT Subpacket

Table 2–6. Contents of ?CLSTAT Subpacket

Offset	Contents
?CLST_AOSVS.PKT_ID (doubleword)	Subpacket identifier. Place the value of ?CLST_AOSVS_PKTID here.
?CLST_AOSVS.SYSTEM (four words)	AOS/VS returns the amount of total system CPU usage.
?CLST_AOSVS.COREM (four words)	AOS/VS returns the amount of core memory manager CPU usage.
?CLST_AOSVS.SYSMGR (four words)	AOS/VS returns the amount of system manager CPU usage.
?CLST_AOSVS.DISK (four words)	AOS/VS returns the amount of disk manager CPU usage.
?CLST_AOSVS. SYSCALLS (four words)	AOS/VS returns the amount of system call CPU usage.
?CLST_AOSVS. SYSCALLS_MOM (four words)	AOS/VS returns the amount of system call CPU usage (mother processor only).
?CLST_AOSVS.DAEMON (four words)	AOS/VS returns the amount of daemon call CPU usage.
?CLST_AOSVS.IDLE (four words)	AOS/VS returns the amount of idle CPU usage.
?CLST_AOSVS.INT (four words)	AOS/VS returns the amount of interrupt CPU usage.
?CLST_AOSVS. RESIDUE (four words)	AOS/VS returns the amount of system residue time.

AOS/VS

?CMATRIX *[packet address]*
error return
normal return

Input

AC0	Reserved (Set to 0.)
AC1	Reserved (Set to 0.)
AC2	Address of the ?CMATRIX packet, unless you specify the address as an argument to ?CMATRIX

Output

AC0	Unchanged
AC1	Unchanged
AC2	Address of the ?CMATRIX packet

Error Codes in AC0

ERICI	Illegal class ID
ERILV	Illegal locality value
ERPRV	Caller not privileged for this action

Why Use It?

Use ?CMATRIX to receive the current class locality scheduling matrix from AOS/VS or to modify this matrix.

Who Can Use It?

For the "get" version of this call, there are no special process privileges needed to issue it, and there are no restrictions concerning file access. For the "set" version of this call, you must have System Manager privilege to issue it, but there are no restrictions concerning file access.

What It Does

The following list describes the format of the class locality scheduling matrix in ?CMATRIX's subpacket.

- The matrix consists of 256 cells, with 2 words in each cell. Each cell contains a user locality, a program locality, and the corresponding class ID.
- User locality values and program locality values appear consecutively in the 2 bytes of the first word in each cell. Both values are between 0 and 15 inclusive.
- The class ID corresponding to a user/program locality value is in the word following the locality value. A class ID value is between 0 and 15 inclusive.

Specifying the "get" code in the main packet has AOS/VS return the class matrix in the subpacket. Specifying the "set" code in the main packet and supplying a class matrix in the subpacket has AOS/VS update its scheduling practices for the indicated users, programs, and classes. However — this updating does not affect processes that are currently running; their classes are *not* modified. When AOS/VS creates processes after ?CMATRIX it gives them their class values according to the new matrix. Use ?LOCALITY to change the class of a process that is currently running.

When you issue ?CMATRIX, indicate as many or as few cells of the matrix as you wish. Here are the rules:

- Place the number of cells (2–word sets) you want to access in the cell count offset of the main packet.
- Supply that many sets of user locality, program locality, and class ID values in the subpacket.
- If you use the "set" version of the call, supply the desired class ID values in the matrix.
- If you use the "set" version of the call and supply duplicate specifications (i.e., multiple values for the same cell), AOS/VS will use the last class ID value given for the cell.
- If you use the "get" version of the call, AOS/VS returns the class IDs associated with the user/program locality pairs that you specified.

Figure 2–9 shows the structure of the ?CMATRIX main packet, and Table 2–7 describes its contents. Figure 2–10 shows the structure of the ?CMATRIX subpacket, and Table 2–8 describes its contents.

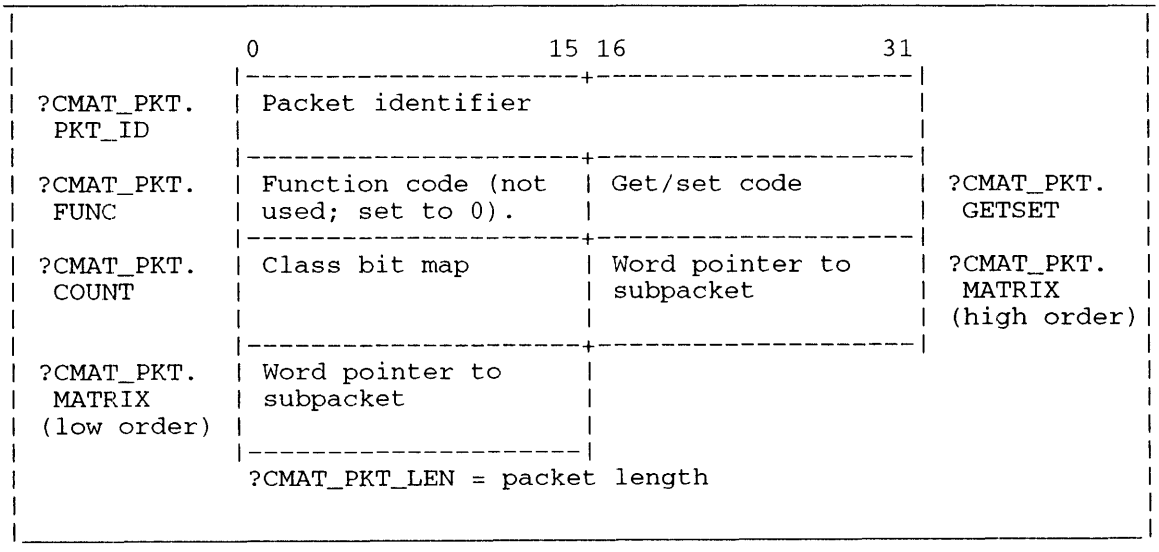


Figure 2–9. Structure of ?CMATRIX Main Packet

Table 2–7. Contents of ?CMATRIX Main Packet

Offset	Contents
?CMAT_PKT.PKT_ID (doubleword)	Packet identifier. Place ?CMAT_PKT_PKTID here.
?CMAT_PKT.FUNC	Function code. Not used (set to 0).
?CMAT_PKT.GETSET	Get/set code. Place the value of ?CMAT_GET here to obtain the class scheduling matrix in the subpacket. Place the value of ?CMAT_SET here to update the matrix; also place the matrix's new values in the subpacket.
?CMAT_PKT.COUNT	The number of cells (2 words in each cell) in the subpacket.
?CMAT_PKT.MATRIX	Word address of the subpacket.

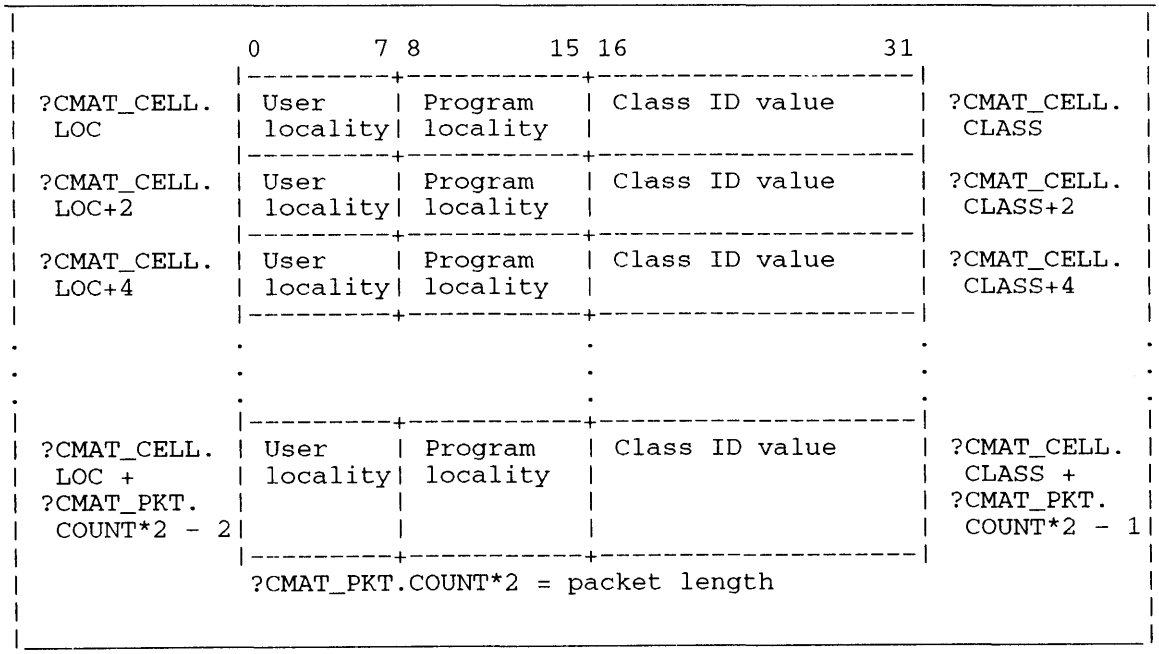


Figure 2–10. Structure of ?CMATRIX Subpacket

Table 2–8. Contents of ?CMATRIX Subpacket

Offset	Contents
?CMAT_CELL.LOC	User locality (left byte) and program locality (right byte) pair for the first cell.
?CMAT_CELL.CLASS	Class ID of the first cell.
?CMAT_CELL.LOC+2	User locality (left byte) and program locality (right byte) pair for the second cell.
?CMAT_CELL.CLASS+2	Class ID of the second cell.
?CMAT_CELL.LOC+4	User locality (left byte) and program locality (right byte) pair for the third cell.
?CMAT_CELL.CLASS+4	Class ID of the third cell.
.	.
.	.
.	.
?CMAT_CELL.LOC + ?CMAT_PKT.COUNT*2-2	User locality (left byte) and program locality (right byte) pair for the last cell.
?CMAT_CELL.CLASS + ?CMAT_PKT.COUNT*2-2	Class ID of the last cell.

It's possible to look at the offsets in Table 2–8 another way. Figure 2–11 shows a relation between the ?CMATRIX main packet and its subpacket.

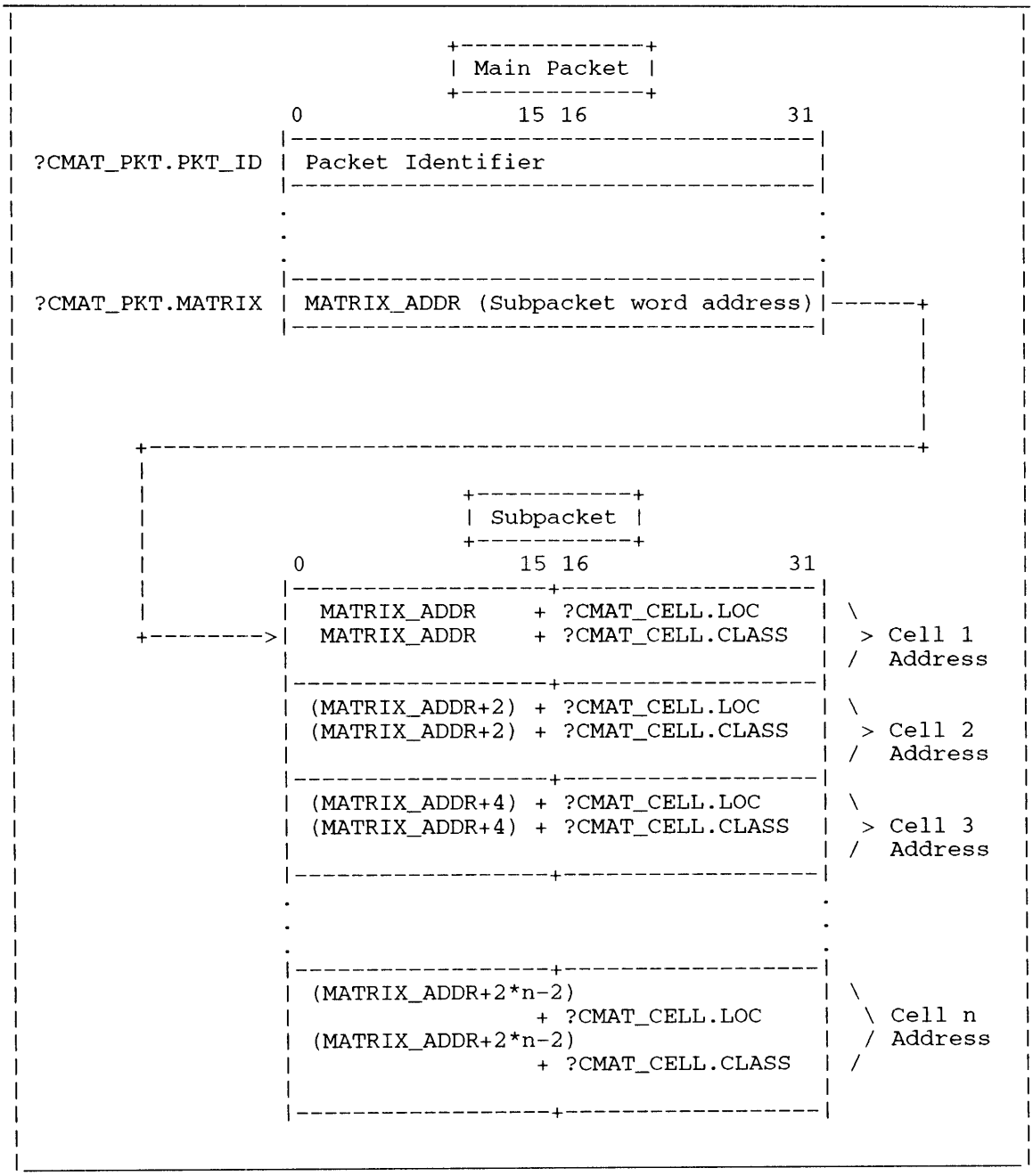


Figure 2–11. Addresses of ?CMATRIX Main Packet and Its Subpacket Offsets

?CON
error return
normal return

Input

AC0 One of the following:

- PID of the server
- Byte pointer to the server's process name

AC1 Bit masks:

?MCPID	if AC0 contains a byte pointer
?MCOBIT	(optional) to suppress the obituary message when the server issues ?DCON, ?RESIGN, or ?CHAIN, or terminates
?MCRNG	if AC2 contains a ring number (If ?MCRNG is not set, the OS assumes Ring 7.)

AC2 Contains the following:

- Bits 0 through 28 are reserved (Set to 0.)
- Bits 29 through 31 contain the server ring number if ?MCRNG is set in AC1

Output

AC0 PID of the server

AC1 Unchanged

AC2 Unchanged

Error Codes in AC0

ERCBK	Connection has been broken
ERCCS	Cannot connect to self (AC0 contains the caller's PID, or a byte pointer to the caller's process name.)
ERNAS	Process is not a server (that is, the target process)
ERPNM	Illegal process name
ERPRH	Attempt to access process not in hierarchy
ERRNI	Ring number invalid
ERVBP	Invalid byte pointer passed as a system call argument

Why Use It?

?CON establishes a connection so that you can use the connection–management facility. The connection–management facility lets you issue the IPC system calls, ?ISEND, ?IREC, and ?IS.R, even though the customer process does not have the ?PVIP privilege.

Who Can Use It?

You need no special process privileges to issue this call. There are no restrictions concerning file access. However, you must connect to a valid server process. After the connection, the server process has access to the memory in the ring from which you issued the call. So you should use ?CON to connect cooperating processes.

What It Does

?CON defines the calling process as a customer of an existing server process that you specify in AC0, and it directs the operating system to create a corresponding connection–table entry. The connection takes place between the caller's PID/ring and a specified server's PID/ring pair.

If the target process is not a declared server (it has not already issued ?SERVE), ?CON fails on error code ERNAS. If the caller tries to connect with itself, ?CON fails and returns error code ERCCS.

If the connection already exists, ?CON resets the status according to the bit mask ?MCOBIT. However, if the customer is trying to reconnect an already broken connection, but the server has not broken it, ?CON fails on error code ERCBK.

When a server disconnects, the operating system sends (by default) an obituary message to the customer. To suppress this message, set ?MCOBIT in AC1.

You cannot connect segment images within the same process.

Notes

- See the description of ?DCON in this chapter.

?CONFIG — Display or reset current MRC routes.

AOS/VS II

?CONFIG *[packet address]*
error return
normal return

Input

AC0	Reserved (set to 0)
AC1	Reserved (set to 0)
AC2	Address of the ?CONFIG packet, unless specified as an argument to ?CONFIG

Output

AC0	Unchanged or an error code
AC1	Unchanged
AC2	Address of the ?CONFIG packet

Error codes returned in AC0

ERPRV	Caller not privileged for this action
ERRVN	Reserved value not zero
ERICD	Illegal function code
ERPKT	Illegal packet id
ERMPR	System call parameter address error
ERPRE	Invalid system call parameter
ERVWP	Invalid address passed as system call argument
ERUNC	Device unknown to host
ERIDT	Illegal device name type

Why Use It?

Use ?CONFIG to either determine the current Message–Based Reliable (MRC) device routes on a system, or to return diverted MRC devices to their primary routes on a specified MRC channel or controller. For MRC devices, primary routes are user–defined and edited in VSGEN, but secondary routes are solely defined by VSGEN and are not user–selectable. A diverted MRC device is one using a secondary route.

Who Can Use it?

Any user who has the access devices privilege.

What It Does

The ?CONFIG system call consists of a main packet and three subpacket functions. The first subpacket function obtains the current route information for a specified MRC device. The other two subpacket functions reset the routes of diverted devices on either a designated channel, or on a controller.

The ?CONFIG_GET CURRENT_ROUTE function returns the current route, the primary route, and the device reset–pending status for a valid MRC device name. The ?CONFIG_RESET_MRC_CHAN function resets the current working routes *back* to their primary routes, if these are available, for all the diverted MRC devices on the designated channel. The ?CONFIG_RESET_MRC_CONTROLLER function resets the current working routes *back* to their primary routes, if these are available, for all the diverted MRC devices on the designated controller.

The ?CONFIG call does not change device routing from primary to secondary routes. Figure 2–11.1 shows the structure of the ?CONFIG main packet.

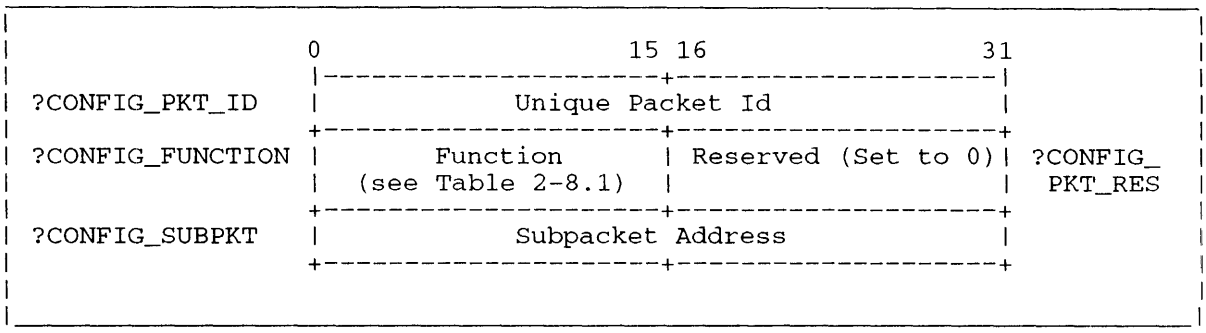


Figure 2–11.1. Structure of ?CONFIG Main Packet

The valid function codes for offset ?CONFIG_FUNCTION are described in Table 2–8.1.

Table 2–8.1. Valid ?CONFIG_FUNCTION Function Codes

Function Code	What It Lets You Do
?CONFIG_GET_CURRENT_ROUTE	Specifies the MRC unit name for the requested route information. The current route, primary route, the unit number, and the reset pending flags are all returned.
?CONFIG_RESET_MRC_CHAN	Specifies the MRC channel's device code for the primary channel requiring resetting. This function attempts to reset all the diverted MRC devices on this channel.
?CONFIG_RESET_MRC_CTRLR	Specifies the MRC channel's device code, and the node number of the primary controller requiring resetting. The channel's device code is the 3-bit IOC, 6-bit device code combination, in which bits 7-9 of the single word contain the IOC number, and bits 10-15 contain the device code. This function attempts to reset all the diverted MRC devices on this controller.

What ?CONFIG_GET_CURRENT_ROUTE Does

The ?CONFIG_GET_CURRENT_ROUTE function accepts as input a byte pointer to the MRC unit name string. ?CONFIG_GET_CURRENT_ROUTE returns the currently active route components, the primary route components, the unit number, and a flag word. The caller must determine whether or not the routes are diverted and require resetting.

The ?CONFIG_GET_CURRENT_ROUTE subpacket structure is shown in Figure 2–11.2, and its contents are described in Table 2–8.2.

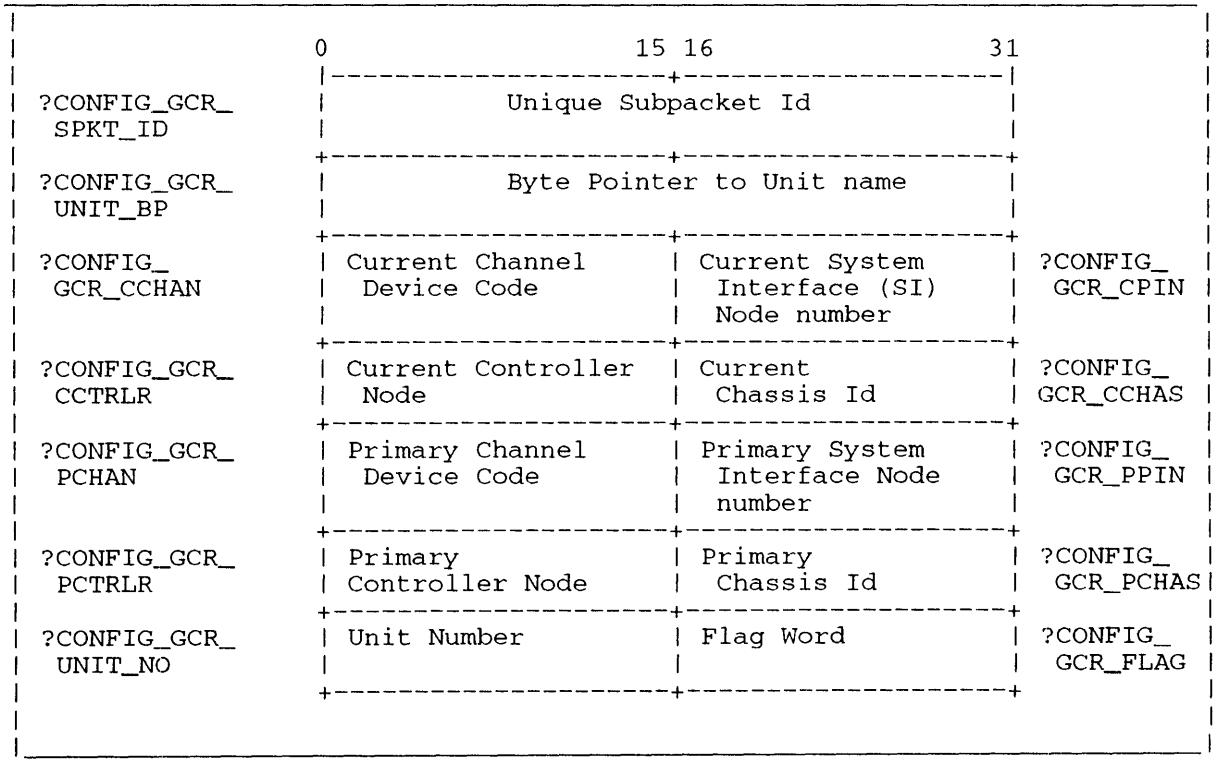


Figure 2–11.2. Structure of the ?CONFIG_GET_CURRENT_ROUTE Function Subpacket

Table 2–8.2. Contents of ?CONFIG_GET_CURRENT_ROUTE Function Subpacket

Offset	Contents
?CONFIG_GCR_SPKT_ID	A unique subpacket identifier (input value).
?CONFIG_GCR_UNIT_BP	A byte pointer to the null terminated unit name string for the MRC device (input value).
?CONFIG_GCR_CCHAN	The channel's device code for the specified unit in the current route (output value).
?CONFIG_GCR_CPIN	The SI node number in the current route to the device (output value).
?CONFIG_GCR_CCTRLR	The controller's node number for the specified unit in the current route (output value).
?CONFIG_GCR_CCHAS	The chassis identifier of the chassis in the current route (output value).
?CONFIG_GCR_PCHAN	The channel's device code for the specified unit in the primary route (output value).
?CONFIG_GCR_PPIN	The SI node number in the primary route to the device (output value).
?CONFIG_GCR_PCTRLR	The controller's node number for the specified unit in the primary route (output value).
?CONFIG_GCR_PCHAS	The primary route chassis identifier (output value).
?CONFIG_GCR_UNIT_NO	The unit number for the specified unit (output value).
?CONFIG_GCR_FLAG	An output flag containing the following bit definitions: ?CONFIG_GCR_RESET_BIT = 0 if no reset operation is pending. ?CONFIG_GCR_RESET_BIT = 1 if there is a reset operation pending. For a disk device, a reset to the new route for the device occurs once the disk is initialized. For tape devices, a reset to the new route occurs when a tape file is next opened on the device.

What ?CONFIG_RESET_MRC_CHAN Does

The ?CONFIG_RESET_MRC_CHAN function accepts as input the device code of the primary channel whose devices require resetting. After hardware repair, resetting the route on the failed channel results in diverted devices on that channel being rerouted to their primary routes.

?CONFIG_RESET_MRC_CHAN immediately tries to reroute all active diverted devices initialized or opened by a ?GOPEN system call. The reset request does not affect current devices using primary routes.

For inactive MRC disk and tape devices, ?CONFIG_RESET_MRC_CHAN sets an internal flag so that the device route is reset when the device is next accessed. The operating system detects the flag and reroutes the device as soon as a newly initialized disk is accessed, or a file is opened on the tape device.

The ?CONFIG_RESET_MRC_CHANNEL subpacket structure is shown in Figure 2–11.3, and its contents are described in Table 2–8.3.

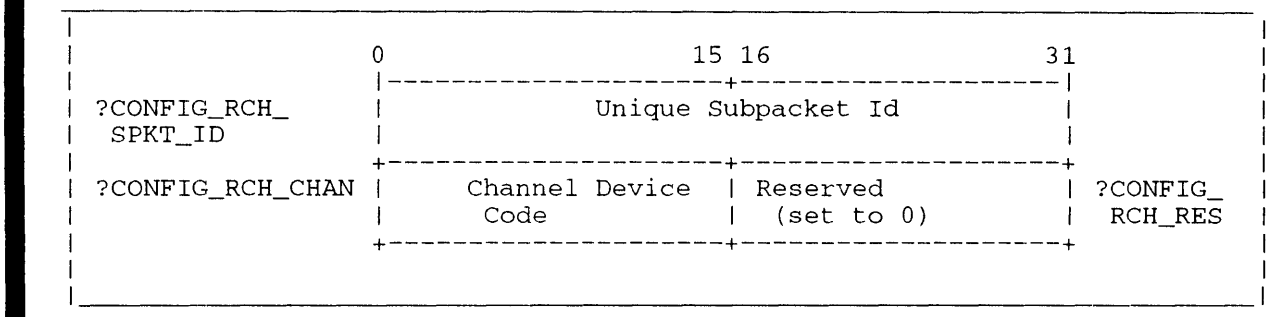


Figure 2–11.3. Structure of the ?CONFIG_RESET_MRC_CHANNEL Function Subpacket

Table 2–8.3 ?CONFIG_RESET_MRD_CHANNEL Function Subpacket Contents

Offset	Contents
?CONFIG_RCH_SPKT_ID	A unique subpacket identifier (an input value).
?CONFIG_RCH_CHAN	The to be reset primary channel's device code (an input value).
?CONFIG_RCH_RES	Reserved (set to 0).

086–000195 updates
093–000542

What ?CONFIG_RESET_MRC_CTRLR does

The ?CONFIG_RESET_MRC_CTRLR function accepts as input the channel's device code, and the primary controller's node number for the controller whose devices require resetting. This function operates the same as the ?CONFIG_RESET_MRC_CHAN function. The channel's device code is required to uniquely identify a controller's node when a configuration contains multiple chassis.

For inactive MRC disk and tape devices, ?CONFIG_RESET_MRC_CTRLR sets an internal flag so that the device route is reset when the device is next accessed. The operating system detects the flag and reroutes the device as soon as a newly initialized disk is accessed, or a file is opened on the tape device.

The ?CONFIG_RESET_MRC_CONTROLLER subpacket structure is shown in Figure 2–11.4, and its contents are described in Table 2–8.4.

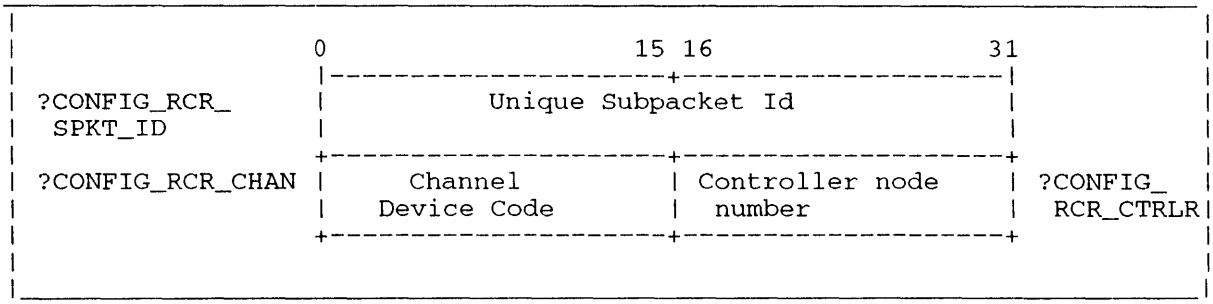


Figure 2–11.4. Structure of the ?CONFIG_RESET_MRC_CTRLR Function Subpacket

Table 2–8.4 ?CONFIG_RESET_MRC_CONTROLLER Function Subpacket Contents

Offset	Contents
?CONFIG_RCR_SPKT_ID	A unique subpacket identifier (an input value).
?CONFIG_RCR_CHAN	Is the channel's device code in the connection between the host and the controller whose devices require resetting (an input value).
?CONFIG_RCR_CTRLR	The primary controller's node number of the controller whose devices require resetting (an input value).

Notes

You can issue ?CONFIG_RESET_MRC_CHAN and ?CONFIG_RESET_MRC_CTRLR calls against a tape device while it is in use, and the call will not affect the active I/O on the tape device. To verify the reset occurred after the subsequent access to the disk or tape device, issue a ?CONFIG with the ?CONFIG_GET_CURRENT_ROUTE subpacket function.

?CONINFO

Request for addressing information on a terminal or console.

AOS/VS II

?CONINFO [packet address]
error return
normal return

Input

AC0	Reserved (set to zero)
AC1	Reserved (set to zero)
AC2	Address of the ?CONINFO main packet, unless specified as an argument to ?CONINFO

Output:

AC0 Unchanged or error code
AC1 Unchanged
AC2 Unchanged

Error codes returned in AC0

ERICN	Illegal channel number
ERIFD	Invalid function for this device
ERIFT	Illegal file type
ERPKT	Illegal packet ID
ERPRE	Illegal system call parameter
ERPRV	Caller not privileged for this action
ERVBP	Invalid byte pointer passed as a system call argument
ERIRB	Insufficient room in buffer
ERVWP	Invalid word pointer passed as a system call argument

Why Use It?

Use ?CONINFO to get console addressing information. Due to the layering of some network protocols, ?CONINFO may only resolve information from the caller's host or the last host that routed the connection. Any console generated by VSGEN in the :PER directory can be verified with ?CONINFO. The console does not have to be active.

Who Can Use It?

The ?CONINFO system call is a 32–bit system call, and is for use only in AOS/VS II. The caller must be PID 2, or have the System Manager privilege turned on, or be the process that owns the console.

What It Does

The ?CONINFO system main packet contains input and output parameters. On input, you provide the unique packet id, a pointer to a buffer for return data, the length of the return data buffer, the channel number of the target console (or a byte pointer to the name of the target console), and a flag specifying which target parameter you supplied.

In addition, if you supply a byte pointer, you must include the length of the console name buffer. The return data buffer must be at least ?CON_UBUF_LEN words long, and is defined in PARU_LONG.SR.

On output, the ?CONINFO main packet lists the amount of data placed in the return buffer. Errors are returned in AC0. The return data buffer contains the requested information.

Figure 2–11.5 contains the ?CONINFO main packet structure, and Table 2–8.5 lists the contents of the ?CONINFO main packet.

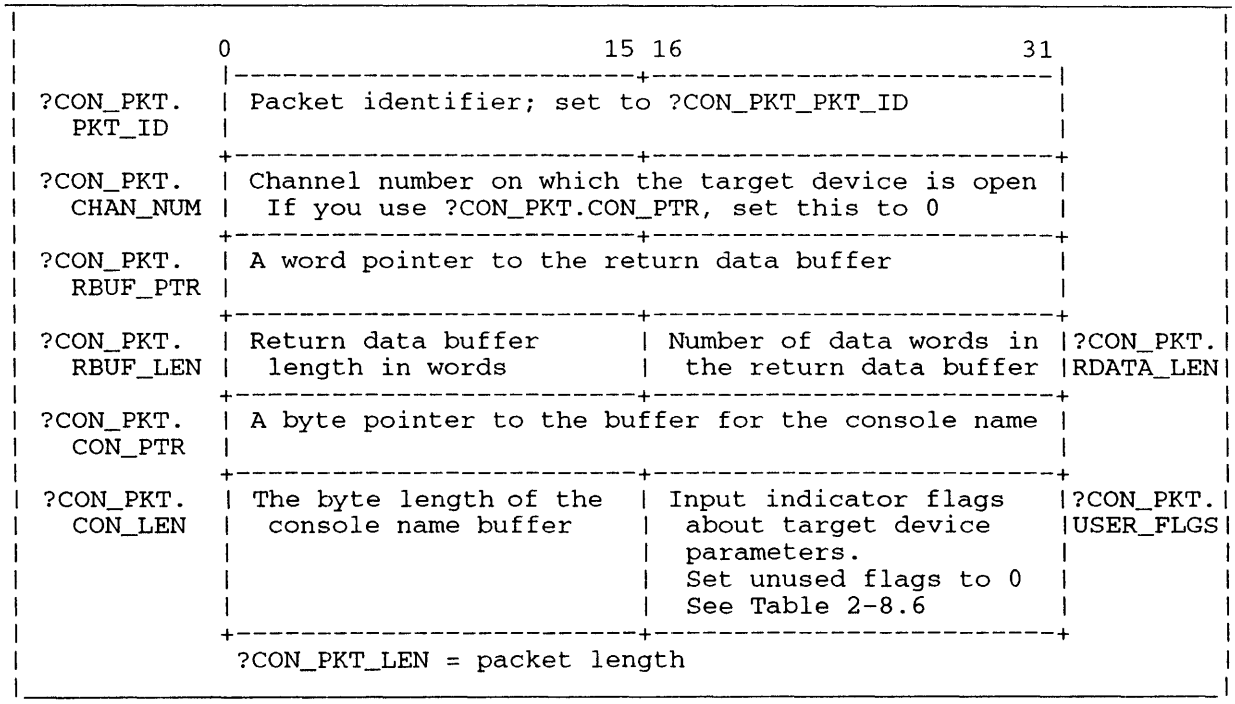


Figure 2–11.5. Structure of ?CONINFO Main Packet

Table 2–8.5. Contents of the ?CONINFO Packet

Offset	Contents
?CON_PKT.PKT_ID	?CON_PKT_PKTID is a unique packet identifier.
?CON_PKT.CHAN_NUM	The channel number on which the target device is open. When using this parameter, ?CON_PKT.CON_PTR and ?CON_PKT.CON_LEN must be zero or an ERPRE error will result.
?CON_PKT.RBUF_PTR	A word pointer to a buffer for return data.
?CON_PKT.RBUF_LEN	The word length of the return data buffer.
?CON_PKT.RDATA_LEN	The number of words of data in the return data buffer.
?CON_PKT.CON_PTR	A byte pointer to the name of the target console. When using this parameter, ?CON_PKT.CHAN_NUM must be set to zero or an ERPRE error will result.
?CON_PKT.CON_LEN	The byte length of the buffer holding the console name. When this parameter is supplied ?CON_PKT.CHAN_NUM must be set to zero or an ERPRE error will result.

Table 2–8.6. Input Values to ?CON_PKT.USER_FLGS Offset

Offset	Contents
?CON_PKT. USER_FLGS	An input flag value indicating that either a byte pointer to the target device name, or the channel number parameter, was supplied. Unused flags must be set to zero or you generate an ERPRE error. The valid flags are: - ?CON_PKT.USER_FLGS.NAME = 1 means the user has supplied a byte pointer to the name of the target device. If this bit is set, the channel number,?CON_CHAN_NUM, must be set to zero. - ?CON_PKT.USER_FLGS.CHAN = 1 means the user supplied the channel number of the target device. If this bit is set, the byte pointer ?CON_CON_PTR, and the length of the name buffer, ?CON_CON_LEN, must both be set to zero.

Return Packet Types

In each return packet in the return data buffer, a value in the first offset (word 0) denotes the type of active connection, or the type of session and session's console. Connection and session types, and their values, are defined in Table 2–8.7.

Return Packets and Line Numbers

Each return packet also contains the Terminal Services (TS) target console line number. In the packet for "soft" controllers, the line numbers are zero relative. Thus, when VSGEN generates "150." Telnet consoles for a Telnet connection across a LAN, the Telnet line numbers range from 0 to 149.

Line numbers for directly connected IAC consoles are relative to each TS engine. Thus when VSGEN generates 128 Intelligent TermController (ITC) TermServer consoles, each of the four TS engines on the ITC owns 32 lines, and each engine's line numbers range from 0 to 31.

Table 2–8.7. ?CON_RET_TYPES Return Buffer Console Types and Definitions

Offset	Description
Word 0	?CON_TCP_RET_TYPE = 1 indicates that the target console is a TermServer console connected through an Intelligent TermController (ITC), or a local-bus terminal controller (LTC) running Transport Control Protocol/Internet Protocol (TCP/IP) software. ?CON_XNS_RET_TYPE = 2 indicates that the target console is a TermServer console connected through an ITC or LTC controller running Xerox Network Services (XNS) software. ?CON_CON_RET_TYPE = 6 indicates that the target console is a CON. These consoles may be Intelligent Asynchronous Controller (IAC) consoles, Ring0 Dual Universal Asynchronous Receiver and Transmitter (DUART) consoles, operator's consoles, or consoles using teletype input and teletype output (TTI-TTO). ?CON_TNET_RET_TYPE = 11 indicates that the target console is a TCP/IP connection (TCON) established over Ring0 Telnet software. ?CON_ITC_MIN_DATA = 14 indicates no network data is available on an ITC/LTC connection, because of some error on the transport engine. ?CON_TSC_MIN_DATA = 15 indicates no network data is available for a ring0 (TCON) network connection. ?CON_PVC_RET_DATA = 16 indicates that the target console is an ITC or LTC controller, and the connection is a permanent virtual circuit (PVC) defined on the controller.

?CON_TCP_RET_TYPE Return Type = 1

AOS/VS II returns the packet for a TermServer connection established through an ITC or an LTC controller using TCP/IP. The TCP/IP packet is shown in Figure 2–11.6, and the packet contents are in Table 2–8.8.

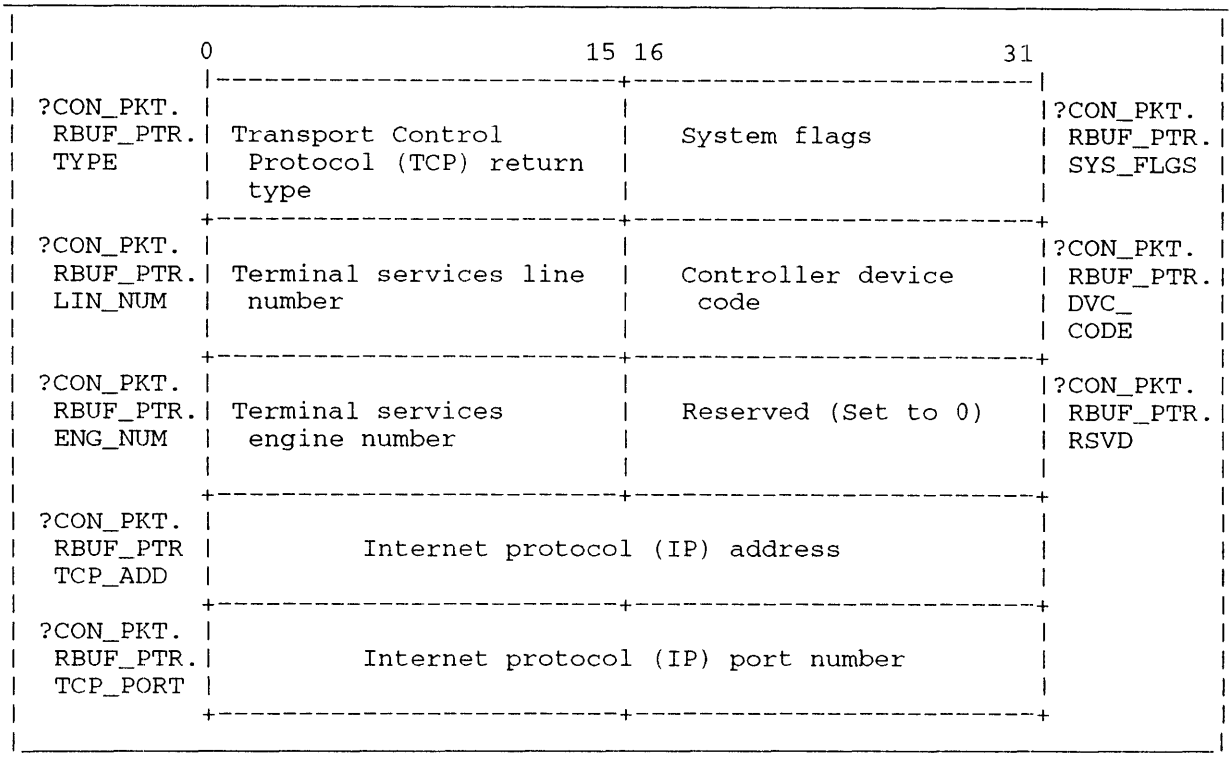


Figure 2–11.6. Structure of ?CON_TCP_RET_TYPE Return Packet

Table 2–8.8. Contents of ?CON_TCP_RET_TYPE Packet

Offset	Contents
?CON_PKT.RBUF_PTR.TYPE	?CON_TCP_RET_TYPE = 1 identifies the type of packet returned (see Table 2-8.7 for a list of types).
?CON_PKT.RBUF_PTR.SYS_FLGS	Valid only for ?CON_CON_RET_TYPE = 6. Otherwise zero is returned.
?CON_PKT.RBUF_PTR.LIN_NUM	The console terminal services line number in decimal.
?CON_PKT.RBUF_PTR.DVC_CODE	The device code of the controller in octal.
?CON_PKT.RBUF_PTR.ENG_NUM	The terminal services controller engine number that owns the line (ranges from 0 to n).
?CON_PKT.RBUF_PTR._RSVD	Reserved. (Set to 0)
?CON_PKT.RBUF_PTR.TCP_ADD	The caller's IP address in decimal. The address is in four contiguous bytes. Each byte contains a decimal triplet from 000 to 255, with the most significant byte on on the left. For example, 107.212.019.036 is an IP address.
?CON_PKT.RBUF_PTR.TCP_PORT	The caller's IP port number in decimal.

?CON_XNS_RET_TYPE Return Type = 2

AOS/VS II returns this packet for a TermServer connection established through an ITC or an LTC controller using Xerox Network Services (XNS).

The packet structure is shown in Figure 2–11.7. Table 2–8.9 describes each offset.

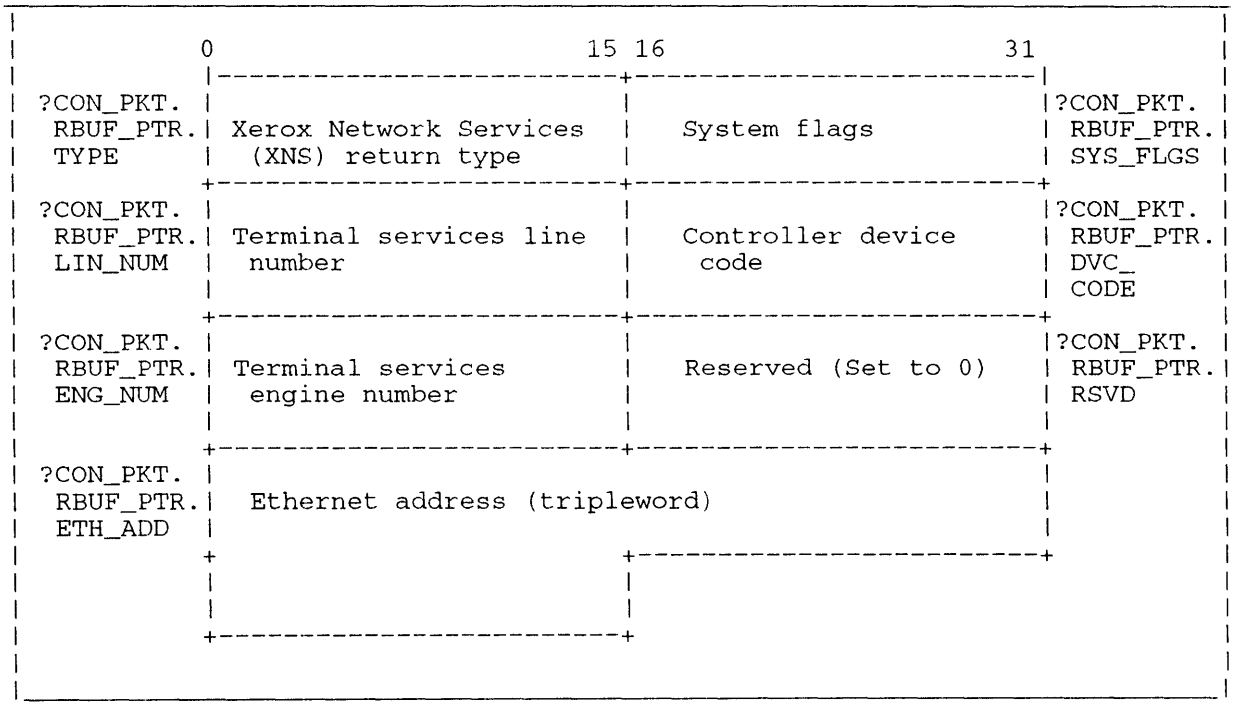


Figure 2–11.7. Structure of ?CON_XNS_RET_TYPE Return Packet

Table 2–8.9. Contents of ?CON_XNS_RET_TYPE Packet

Offset	Contents
?CON_PKT.RBUF_PTR.TYPE	?CON_XNS_RET_TYPE = 2 identifies the type of packet returned (see Table 2-8.7 for a list of types).
?CON_PKT.RBUF_PTR.SYS_FLGS	Valid only for ?CON_CON_RET_TYPE = 6. Otherwise zero is returned.
?CON_PKT.RBUF_PTR.LIN_NUM	The console terminal services line number in decimal.
?CON_PKT.RBUF_PTR.DVC_CODE	The device code of the TermServer controller in octal.
?CON_PKT.RBUF_PTR.ENG_NUM	The terminal services controller engine number that owns the line (ranges from 0 to n).
?CON_PKT.RBUF_PTR.RSVD	Reserved. (Set to 0)
?CON_PKT.RBUF_PTR.ETH_ADD (tripleword)	The Ethernet address of the TermServer box--12 hexadecimal digits.

?CON_CON_RET_TYPE Return Type = 6

AOS/VS II returns this packet for consoles directly connected to IACs, modem lines, Ring0 DUARTS, Opcons, and consoles using TTI–TTO. The packet structure is shown in Figure 2–11.8. Table 2–8.10 describes each offset.

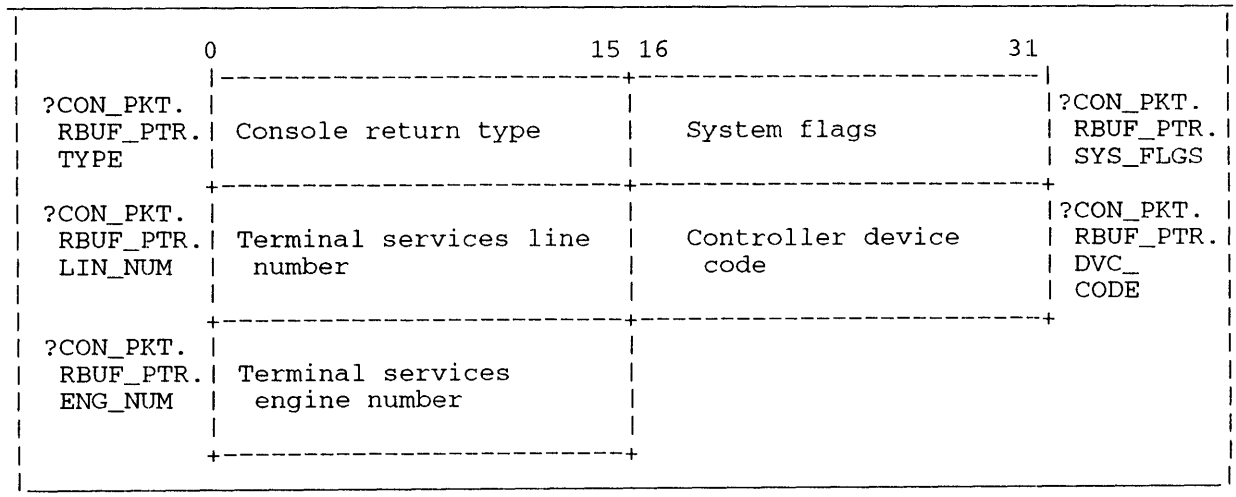


Figure 2–11.8. Structure of ?CON_CON_RET_TYPE Return Packet

Table 2–8.10. Contents of ?CON_CON_RET_TYPE Packet

Offset	Contents
?CON_PKT.RBUF_PTR.TYPE	?CON_CON_RET_TYPE = 6 identifies the type of packet returned (see Table 2-8.7 for a list of types).
?CON_PKT.RBUF_PTR.SYS_FLGS	Returns one of the following values: ?CON_PKT.RBUF_PTR.SYS_FLGS.OPCON = 1 if the target console is the system operator's console. ?CON_PKT.RBUF_PTR.SYS_FLGS.MODEM = 1 if the target console's modem characteristic is switched on.
?CON_PKT.RBUF_PTR.LIN_NUM	The terminal services console line number in decimal.
?CON_PKT.RBUF_PTR.DVC_CODE	The controller device codes, for example DUART = 34 (octal) TTI-TTO = 11 (octal) IAC = 41 (octal)
?CON_PKT.RBUF_PTR.ENG_NUM	The terminal services controller engine number that owns the line (ranges from 0 to n).

?CON_TNET_RET_TYPE Return Type = 11

The Ring0 Telnet connection (TCON) return packet is shown in Figure 2–11.9, and the packet contents are in Table 2–8.11.

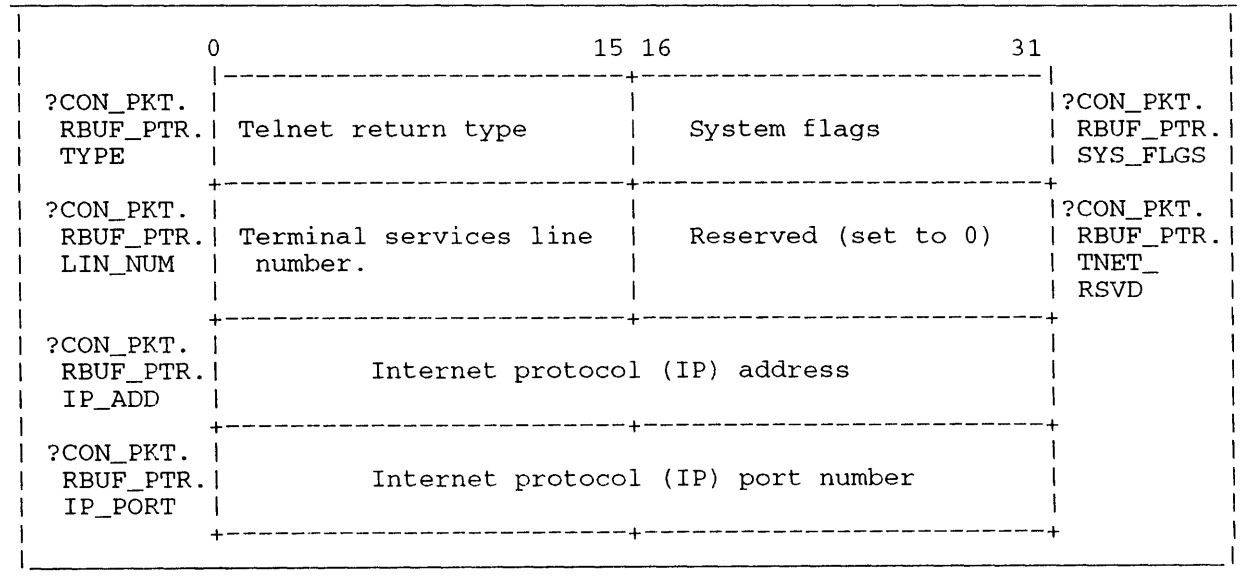


Figure 2–11.9. Structure of ?CON_TNET_RET_TYPE Return Packet

Table 2–8.11. Contents of ?CON_TNET_RET_TYPE Return Packet

Offset	Contents
?CON_PKT.RBUF_PTR.TYPE	?CON_TNET_RET_TYPE = 11 identifies the type of packet returned (see Table 2-8.7 for a list of types.)
?CON_PKT.RBUF_PTR.SYS_FLGS	Valid only for ?CON_CON_RET_TYPE = 6. Otherwise zero is returned.
?CON_PKT.RBUF_PTR.LIN_NUM	The console terminal services line number in decimal.
?CON_PKT.RBUF_PTR.TNET_RSVD	Reserved. (Set to 0)
?CON_PKT.RBUF_PTR.IP_ADD	The caller's IP address in decimal. (see page NO TAG for a description).
?CON_PKT.RBUF_PTR.IP_PORT	The caller's IP port number in decimal.

?CON_ITC_MIN_DATA Return Type = 14

AOS/VS II returns the ?CON_ITC_MIN_DATA packet for a TermServer console established through an ITC or an LTC controller using XNS or TCP when no network connection data is available. The condition can occur when there is no connection on the TermServer line, or when there is a connection, but the transport created the connection without any ?CONINFO data for the line. Revision incompatibility between TS software and your TermServer network software can also produce this condition.

The ?CON_ITC_MIN_DATA packet structure is shown in Figure 2–11.10. Table 2–8.12 describes each offset.

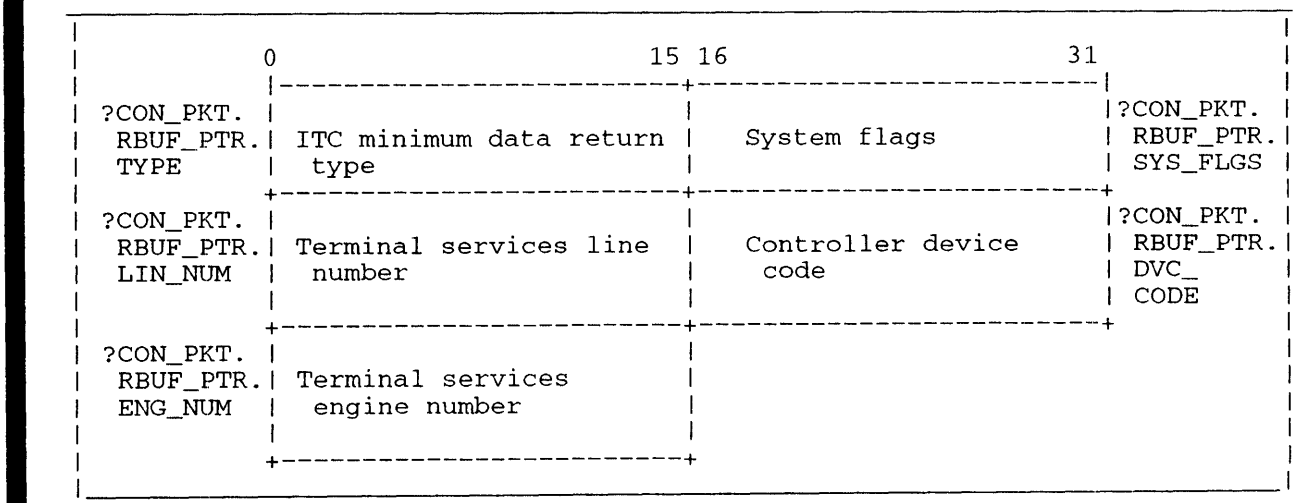


Figure 2–11.10. Structure of ?CON_ITC_MIN_DATA Return Packet

Table 2–8.12. Contents of ?CON_ITC_MIN_DATA Packet

Offset	Contents
?CON_PKT.RBUF_PTR.TYPE	?CON_ITC_MIN_DATA = 14 identifies the type of packet returned (see Table 2-8.7 for a list of types).
?CON_PKT.RBUF_PTR.SYS_FLGS	Valid only for ?CON_CON_RET_TYPE = 6. Otherwise zero is returned.
?CON_PKT.RBUF_PTR.LIN_NUM	The console terminal services line number in decimal.
?CON_PKT.RBUF_PTR.DVC_CODE	The device code of the TermServer controller in octal.
?CON_PKT.RBUF_PTR.ENG_NUM	The terminal services controller engine number that owns the line (ranges from 0 to n).

?CON_TSC_MIN_DATA Return Type = 15

AOS/VS II returns the ?CON_TSC_MIN_DATA packet for a Ring0 "soft" controller (TCON) when there is no connection on the console and no network information available.

The ?CON_TSC_MIN_DATA packet structure is shown in Figure 2–11.11, and Table 2–8.13 describes each offset.

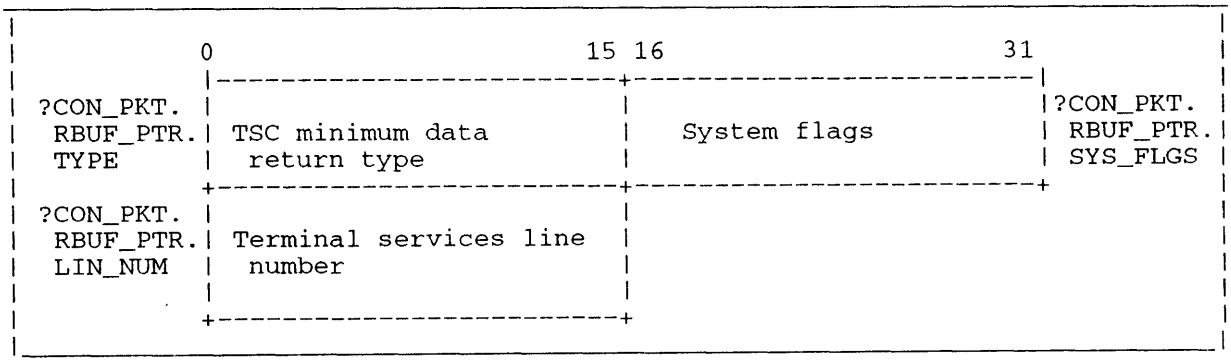


Figure 2–11.11. Structure of ?CON_TSC_MIN_DATA Return Packet

Table 2–8.13. Contents of ?CON_TSC_MIN_DATA Packet

Offset	Contents
?CON_PKT.RBUF_PTR.TYPE	?CON_TSC_MIN_DATA = 15 identifies the type of packet returned (see Table 2-8.7 for a list of types).
?CON_PKT.RBUF_PTR.SYS_FLGS	Valid only for ?CON_CON_RET_TYPE = 6. Otherwise zero is returned.
?CON_PKT.RBUF_PTR.LIN_NUM	The console terminal services line number in decimal.

?CON_PVC_RET_TYPE Return Type = 16

AOS/VS II returns the ?CON_PVC_RET_TYPE packet for a TermServer connection established through an ITC or an LTC controller using PVCs over either the XNS or TCP protocols. The packet contains the controller device code, the number of the TS engine owning the line, and the line number on the engine that owns the console. The packet also contains a PVC subtype field describing the format of the data defining the PVC connection address.

The packet's first seven words are common to all PVC packets. The ?CON_PVC_RET_TYPE packet structure is shown in Figure 2–11.12, and Table 2–8.14 describes each offset.

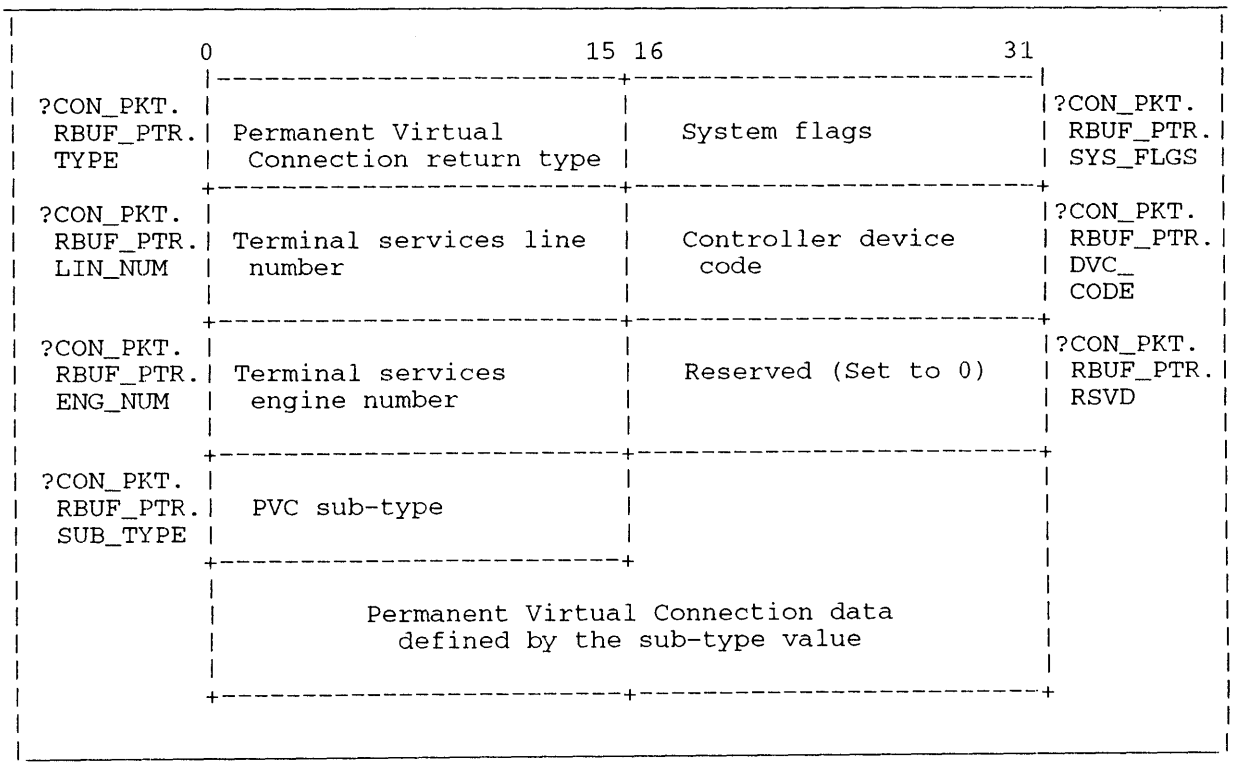


Figure 2–11.12. Structure of ?CON_PVC_RET_TYPE Return Packet

Table 2–8.14. Contents of ?CON_PVC_RET_TYPE Packet

Offset	Contents
?CON_PKT.RBUF_PTR.TYPE	?CON_PVC_RET_TYPE = 16 identifies the type of packet returned (see Table 2-8.7 for a list of types).
?CON_PKT.RBUF_PTR.SYS_FLGS	Valid only for ?CON_CON_RET_TYPE = 6. Otherwise zero is returned.
?CON_PKT.RBUF_PTR.LIN_NUM	The console terminal services line number in decimal.
?CON_PKT.RBUF_PTR.DVC_CODE	The device code of the controller.
?CON_PKT.RBUF_PTR.ENG_NUM	The terminal services controller engine number that owns the line (ranges from 0 to n).
?CON_PKT.RBUF_PTR.RSVD	Reserved. (Set to 0)
?CON_PKT.RBUF_PTR.SUB_TYPE	A value in the subpacket defining one of seven different types of data returned from the ITC describing the PVC connection address. The values are: ?CON_PVC_NAME = 0 ?CON_PVC_NAME_PORT = 1 ?CON_PVC_IP = 2 ?CON_PVC_IP_PORT = 3 ?CON_PVC_ETH = 4 ?CON_PVC_PORT = 5 ?CON_PVC_NET = 6

?CON_PVC_RET_TYPE Subpackets

The ?CON_PVC_RET_TYPE packet returns a PVC subtype and its corresponding subpacket. The first six words of the subpacket are identical to ?CON_PVC_RET_TYPE. The subtypes begin at Word 7 in each subpacket. In the following sections, the PVC–specific content of the subpackets is enumerated in PVC subtype order.

The subtypes and their values, are defined in Table 2–8.15.

Table 2–8.15. ?CON_PVC_RET_TYPE Subpacket Types and Definitions

Offset	Definitions
?CON_PKT.RBUF_PTR.SUB_TYPE	?CON_PVC_NAME = 0 is the host name upon which the TermServer console connection is established through an ITC or an LTC controller using PVCs over either an XNS or TCP protocol. The connection's host name is returned. ?CON_PVC_NAME_PORT = 1 is a TermServer connection established through an ITC or an LTC controller using PVCs over an XNS or TCP protocol. The connection's host name and port are returned. ?CON_PVC_IP = 2 is a TermServer connection established through an ITC or an LTC controller using PVCs over TCP protocol. The console's IP address is returned. ?CON_PVC_IP_PORT = 3 is a TermServer connection established through an ITC or an LTC controller using PVCs over TCP protocol. The console's IP address and port are returned. ?CON_PVC_ETH = 4 is a TermServer connection established through an ITC or an LTC controller using PVCs over XNS protocol. The console's Ethernet address is returned. ?CON_PVC_PORT = 5 is a TermServer connection established through an ITC or an LTC controller using PVCs over XNS protocol. The console's port number is returned. ?CON_PVC_NET = 6 is a TermServer connection established through an ITC or an LTC controller using PVCs over XNS protocol. The console's network address, port number, and Ethernet address is returned.

?CON_PVC_NAME Subtype = 0 — The packet contains the first six words as described in Figure 2–11.12 on page 2–58.16. The PVC_specific data in the ?CON_PVC_NAME packet structure are shown in Figure 2–11.13, and Table 2–8.16 describes each offset.

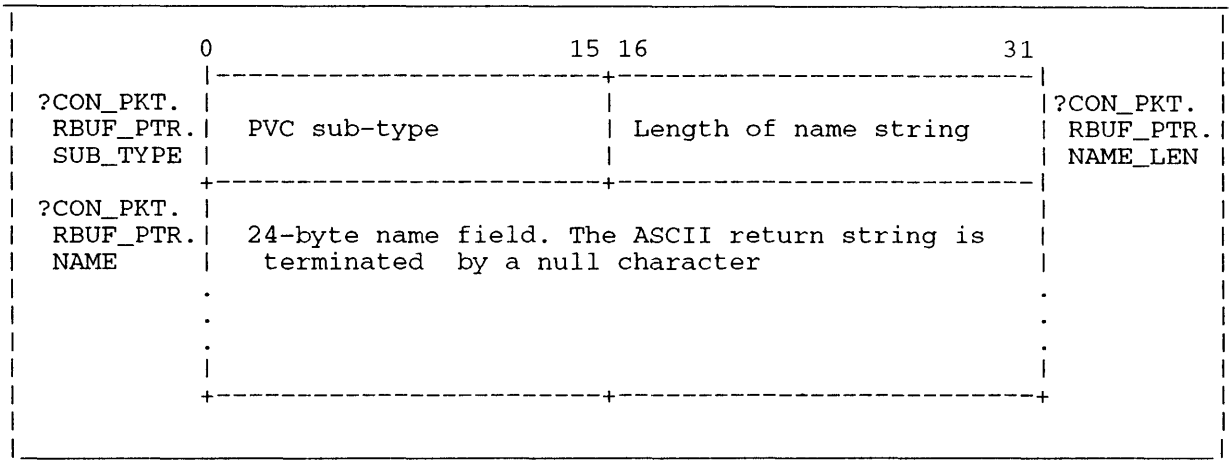


Figure 2–11.13. Structure of ?CON_PVC_NAME Return Packet

Table 2–8.16. Contents of ?CON_PVC_NAME Subpacket

Offset	Contents
?CON_PKT.RBUF_PTR.SUB_TYPE	?CON_PVC_NAME = 0 identifies the type of PVC data (see Table 2-8.15 for a list of types.)
?CON_PKT.RBUF_PTR.NAME_LEN	The length of the ASCII name string in ?CON_PKT.RBUF_PTR.NAME in bytes.
?CON_PKT.RBUF_PTR.NAME	The name bound to an XNS or TCP physical address for the connection. The name is a null-terminated ASCII string with a maximum length of 24 bytes.

?CON_PVC_NAME_PORT Subtype = 1 — The packet contains the first six words as described in Figure 2–11.12 on page 2–58.16. The PVC–specific data, shown in the ?CON_PVC_NAME_PORT packet structure in Figure 2–11.14. Each offset is described in Table 2–8.17.

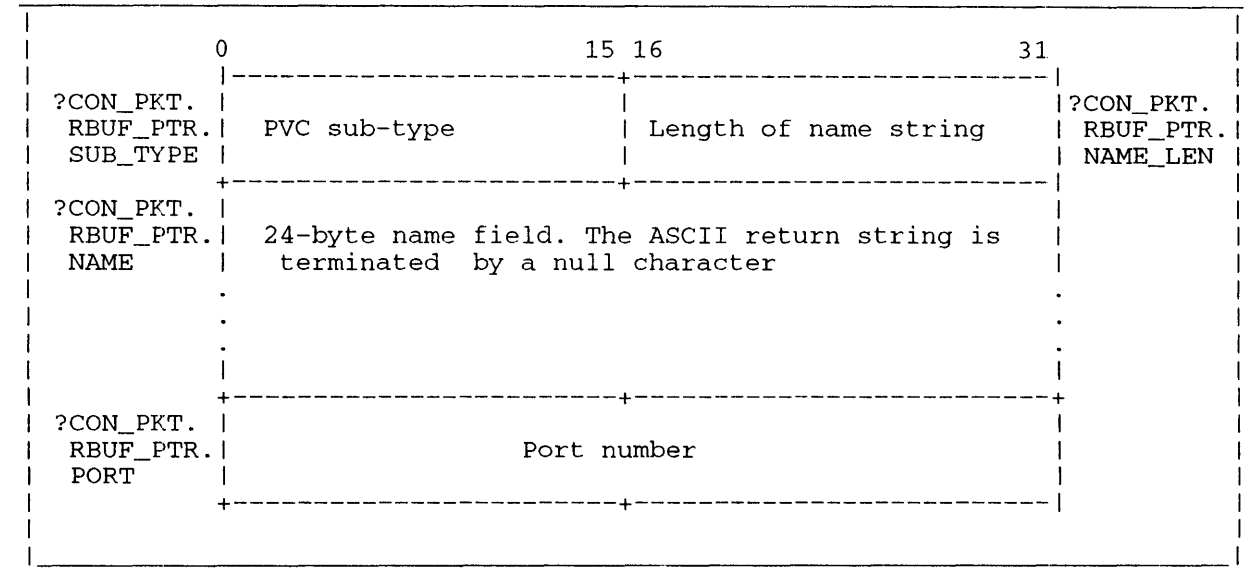


Figure 2–11.14 Structure of ?CON_PVC_NAME_PORT return packet

Table 2–8.17. Contents of ?CON_PVC_NAME_PORT Subpacket

Offset	Contents
?CON_PKT.RBUF_PTR.SUB_TYPE	?CON_PVC_NAME_PORT = 1 identifies the type of PVC data (see Table 2-8.15 for a list of types).
?CON_PKT.RBUF_PTR.NAME_LEN	The length of the ASCII name string in ?CON_PKT.RBUF_PTR.NAME in bytes.
?CON_PKT.RBUF_PTR.NAME	The name bound to an XNS or TCP physical address for the connection. The name is a null terminated ASCII string with a maximum length of 24 bytes.
?CON_PKT.RBUF_PTR.PORT	The port number on the host supporting the TCP physical connection.

?CON_PVC_IP Subtype = 2 — The packet contains the first six words as described in Figure 2–11.12 on page 2–58.16. The PVC–specific data in the ?CON_PVC_IP packet structure are shown in Figure 2–11.15. Each offset is described in Table 2–8.18.

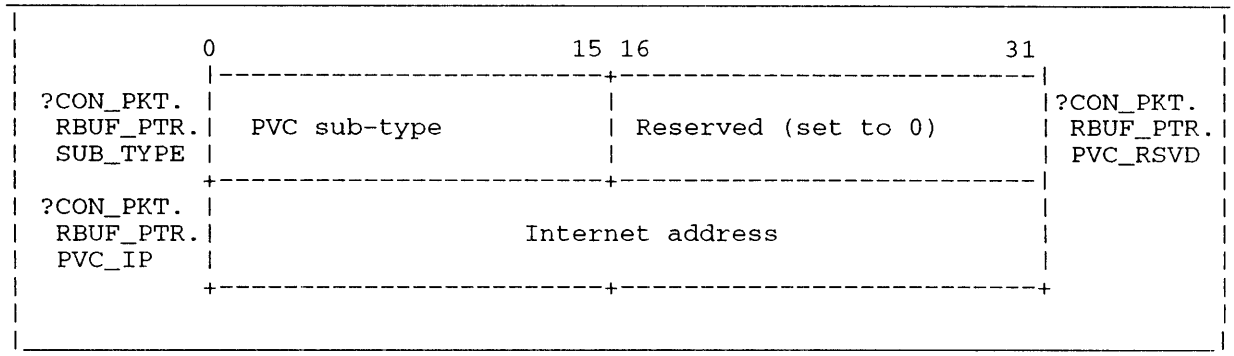


Figure 2–11.15. Structure of ?CON_PVC_IP return packet

Table 2–8.18. Contents of ?CON_PVC_IP Subpacket

Offset	Contents
?CON_PKT.RBUF_PTR.SUB_TYPE	?CON_PVC_IP = 2 identifies the type of PVC data (see Table 2-8.15 for a list of types).
?CON_PKT.RBUF_PTR.PVC_RSVD	Reserved. Set to 0.
?CON_PKT.RBUF_PTR.PVC_IP	The Internet address of the PVC connection. Returned to the packet in four contiguous decimal bytes (See Table 2-8.8 for an explanation).

?CON_PVC_IP_PORT Subtype = 3 — The packet contains the first six words as described in Figure 2–11.12 on page 2–58.16. The PVC_specific data in the ?CON_PVC_IP_PORT packet structure are shown in Figure 2–11.16. Each offset is described in Table 2–8.19.

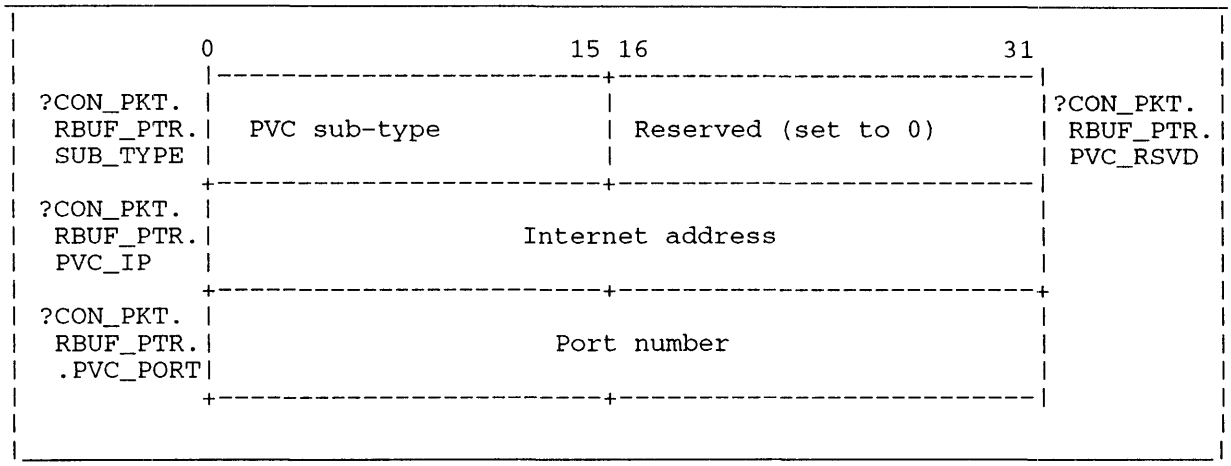


Figure 2–11.16 Structure of ?CON_PVC_IP_PORT return packet

Table 2–8.19. Contents of ?CON_PVC_IP_PORT Subpacket

Offset	Contents
?CON_PKT.RBUF_PTR.SUB_TYPE	?CON_PVC_IP_PORT = 3 identifies the type of PVC data (see Table 2-8.15 for a list of types).
?CON_PKT.RBUF_PTR.PVC_RSVD	Reserved. Set to 0.
?CON_PKT.RBUF_PTR.PVC_IP	The Internet address of the PVC connection. Returned to the packet in four contiguous hexadecimal bytes (See Table 2-8.8 for an explanation).
?CON_PKT.RBUF_PTR.PVC_PORT	The Internet port number of the PVC connection.

?CON_PVC_ETH Subtype = 4 — The packet contains the first six words as described in Figure 2–11.12 on page 2–58.16. The PVC–specific data in the ?CON_PVC_ETH packet structure are shown in Figure 2–11.17. Each offset is described in Table 2–8.20.

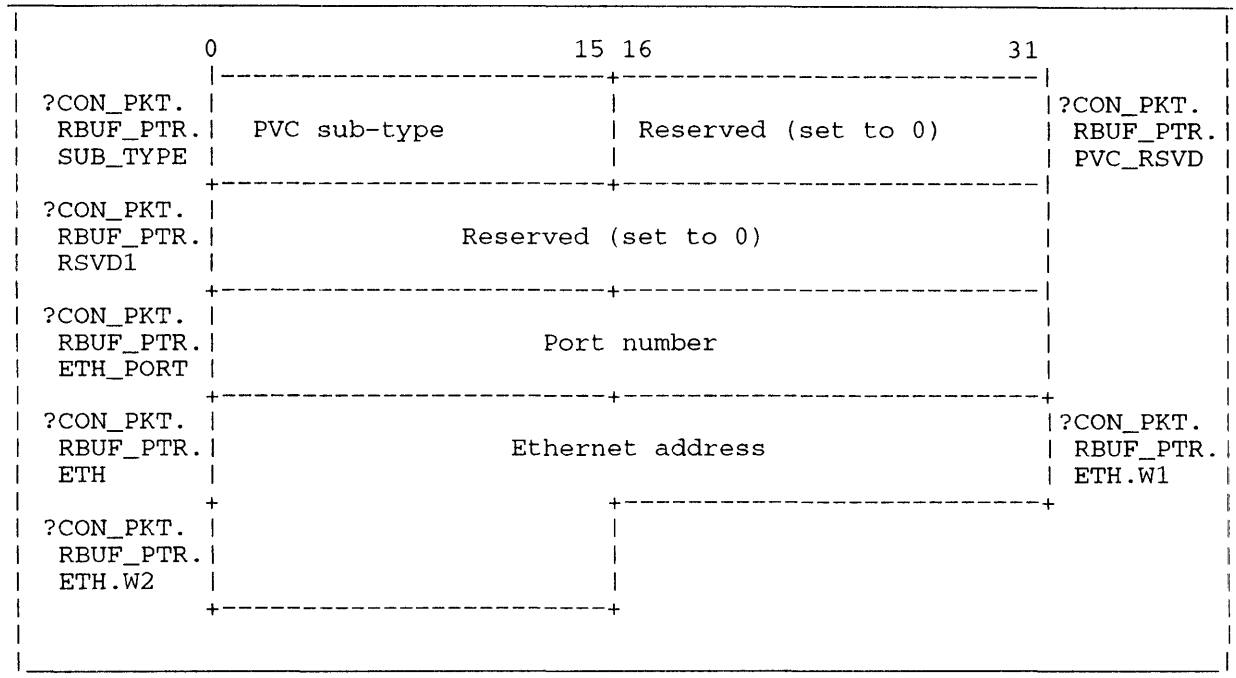


Figure 2–11.17 Structure of ?CON_PVC_ETH return packet

Table 2–8.20. Contents of ?CON_PVC_ETH Subpacket

Offset	Contents
?CON_PKT.RBUF_PTR.SUB_TYPE	?CON_PVC_ETH = 4 identifies the type of PVC data (see Table 2-8.15 for a list of types).
?CON_PKT.RBUF_PTR.PVC_RSVD	Reserved. Set to 0.
?CON_PKT.RBUF_PTR.PVC_RSVD1	Reserved. Set to 0.
?CON_PKT.RBUF_PTR.ETH_PORT	The Ethernet port number of the PVC connection.
?CON_PKT.RBUF_PTR.ETH	The Ethernet address of the PVC connection, returned to the packet in 12 contiguous hexadecimal digits.

?CON_PVC_PORT Subtype = 5 — The packet contains the first six words as described in Figure 2–11.12 on page 2–58.16. The PVC–specific data in the ?CON_PVC_PORT packet structure are shown in Figure 2–11.18. Each offset is described in Table 2–8.21.

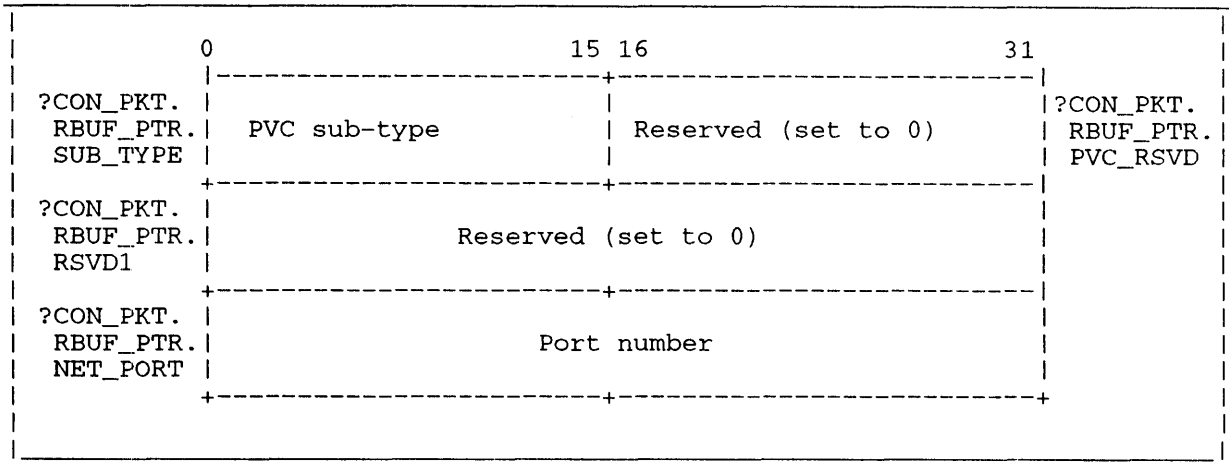


Figure 2–11.18 Structure of ?CON_PVC_PORT return packet

Table 2–8.21. Contents of ?CON_PVC_PORT Subpacket

Offset	Contents
?CON_PKT.RBUF_PTR.SUB_TYPE	?CON_PVC_ETH = 4 identifies the type of PVC data. (See Table 2-8.15 for a list of types.)
?CON_PKT.RBUF_PTR.PVC_RSVD	Reserved. Set to 0.
?CON_PKT.RBUF_PTR.PVC_RSVD1	Reserved. Set to 0.
?CON_PKT.RBUF_PTR.NET_PORT	The Ethernet port number of the PVC connection.

?CON_PVC_NET Subtype = 6 — The packet contains the first six words as described in Figure 2–11.12 on page 2–58.16. The PVC–specific data in the ?CON_PVC_NET packet structure are shown in Figure 2–11.19. Each offset is described in Table 2–8.22.

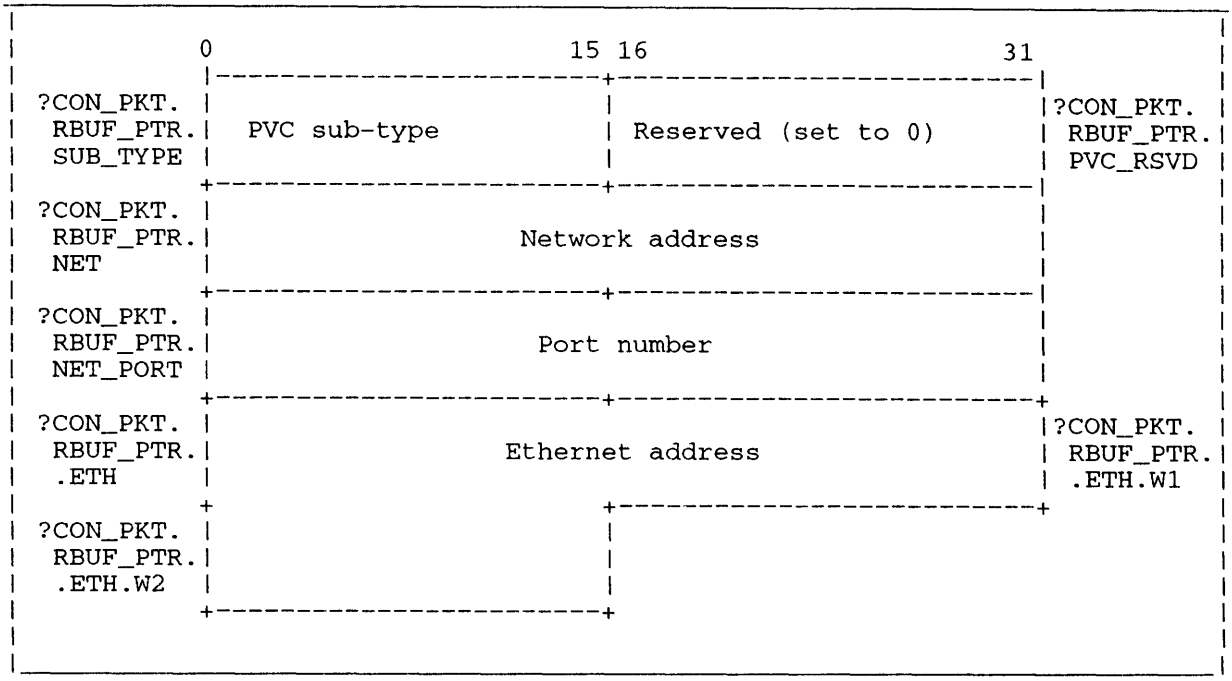


Figure 2–11.19 Structure of ?CON_PVC_NET return packet

Table 2–8.22. Contents of ?CON_PVC_NET Subpacket

Offset	Contents
?CON_PKT.RBUF_PTR.SUB_TYPE	?CON_PVC_NET = 6 identifies the type of PVC data (see Table 2–8.15 for a list of types).
?CON_PKT.RBUF_PTR.PVC_RSVD	Reserved. Set to 0.
?CON_PKT.RBUF_PTR.NET	The Network address of the PVC connection. Returned to the packet in 8 contiguous hexadecimal digits.
?CON_PKT.RBUF_PTR.NET_PORT	The Ethernet port number of the PVC connection.
?CON_PKT.RBUF_PTR.NET.ETH	The Ethernet address of the PVC connection, returned to the packet in 12 contiguous hexadecimal digits.

?CPMAX Sets maximum size for a control point directory (CPD).

?CPMAX *[packet address]*
error return
normal return

Input

AC0 One of the following:

- Byte pointer to the CPD's pathname
- 0 if a packet address is supplied

AC1 One of the following:

- CPD's new maximum space (MS) value if not supplying a packet
- Unused if supplying a packet

AC2 One of the following:

- Reserved (Set to 0 if not supplying a packet.)
- Address of the ?CPMAX packet, unless you specify the address as an argument to ?CPMAX

Output

AC0 Unchanged

AC1 Unchanged

AC2 Unchanged

Error Codes in AC0

ERCPD CPD maximum size exceeded
ERIFT Illegal file type (The target is not a CPD or is the root.)
ERVBP Invalid byte pointer passed as a system call argument
ERWAD Write access denied
ER_FS_DIRECTORY_NOT_AVAILABLE
Directory not available because the file system is force released (AOS/VS II only)
ER_FS_TLA_MODIFY_VIOLATION
Attempt to modify an AOS/VS II file with ?ODTL value supplied in ?GOPEN packet

Why Use It?

?CPMAX lets you restrict the space that is assigned to a CPD and its subordinate directories to a predefined limit. This is useful for managing and/or conserving your system's disk space.

Who Can Use It?

There are no special process privileges needed to issue this call. The calling process must have Write access to the target CPD.

What It Does

?CPMAX changes the maximum space (MS) value of the target CPD. The target CPD can be specified in one of two ways: either by a byte pointer to the CPD's pathname in AC0, or by using

offset ?CPMCN in the ?CPMAX packet. The use of a packet is only necessary if you choose to specify the target CPD's channel number. AC0 must be set to zero to indicate the use of a packet.

If a packet is not being used, then AC1 contains the CPD's new maximum space value. If a packet is being used, offsets ?CPMHS and ?CPMLS must be used to indicate the new MS value and AC1 is unused.

A CPD's MS value defines the maximum number of disk blocks available to the CPD and all of its subordinate files, except for files that are part of a subordinate logical disk (LD).

You cannot set a CPD's MS value so that its current space value (the number of blocks currently allocated) would exceed the MS value of a superior control point directory, because ?CPMAX will fail on error ERCPD.

Figure 2–12 shows the structure of the ?CPMAX packet.

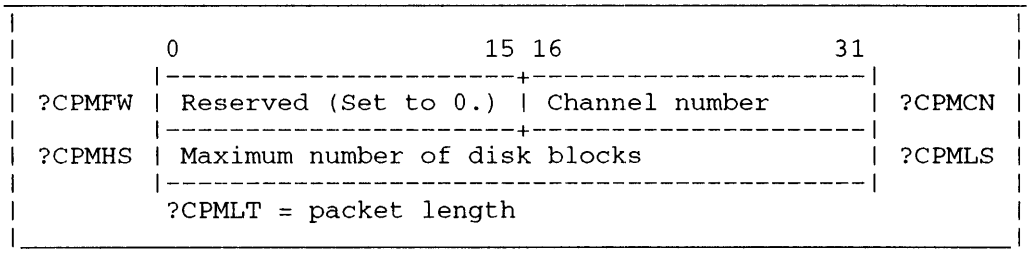


Figure 2–12. Structure of ?CPMAX packet

?CREATE — Creates a file or directory.

?CREATE *[packet address]*
error return
normal return

Input

AC0	Byte pointer to a pathname for the new file
AC1	Reserved (Set to 0.)
AC2	Address of the ?CREATE packet, unless you specify the address as an argument to ?CREATE

Output

AC0	Unchanged
AC1	Undefined
AC2	Address of the ?CREATE packet

Error Codes in AC0

ERIFT	Illegal file type
ERIVP	Invalid port number specified
ERLVL	Maximum directory tree depth exceeded
ERMPR	System call parameter address error
ERNAE	Filename already exists
ERVBP	Invalid byte pointer passed as a system call argument
ERWAD	Write access denied
ERWBP	Invalid word pointer passed as a system call argument

Why Use It?

?CREATE lets you create a file or directory for I/O. (As an alternative, you can create and open a file simultaneously by selecting the creation option in the ?OPEN packet.) ?CREATE is also useful for creating files to serve as ports for IPC messages.

Who Can Use It?

There are no special process privileges needed to issue this call. You must have Execute access and either Write or Append access to the target file's directory.

What It Does

You can use ?CREATE to create files of nearly any type, except those that represent certain peripheral devices. (See Table 2–9 for a complete list of the types of files that you can create with ?CREATE.)

To create a file for the peripheral directory (:PER), the ?CREATE caller must have Write access to the peripheral directory. Otherwise, ?CREATE takes the normal return, but has no effect.

Table 2–9. Valid ?CREATE File Types

Type	Meaning	Comments
?FUDF	User Data File	Usually applies to object files.
?FTXT	Text File	Should contain ASCII text.
?FPRG	AOS Program File	Program file for use under AOS (16-bit code).
?FUNX	MV/UX File	File for use under MV/UX.
?FPRV	AOS/VS Program File	Program file for use under AOS/VS (32- or 16-bit code).
?FDIR	Disk Directory	None.
?FCPD	Control Point Directory	None.
?FLNK	Link File	None.
?FSTF	Symbol Table File	Produced by the Link utility and used primarily by the OS.
?FUPF	User Profile File	Used by PREDITOR (user profile editor) and EXEC.
?FSDF	System Data File	You cannot specify record format types (left byte of offset ?CFTYP) for system data files.
?FIPC	IPC Port Entry	IPC file.
?FGLT	Generic Labeled Tape	None.
?FSPR	Spoolable Peripheral Directory	None.
?FQUE	Queue Entry	None.
?FNCC ?FLCC ?FFCC ?FOCC	FORTRAN Carriage Control	None.
?FPIP	Pipe File	The element size must be a multiple of 4.

Before you issue ?CREATE, you must set up the ?CREATE packet in your address space. You can load the packet address into AC2 before you issue ?CREATE, or you can cite the packet address as an argument to ?CREATE. You must define a pathname for the file, and load AC0 with a byte pointer to that pathname.

To create a link entry with ?CREATE, specify the link name as the last entry in the pathname. Also, place a byte pointer to the link resolution pathname in offset ?CACP of the packet. When you create a link, the operating system does not resolve the pathname associated with the link. Therefore, if the pathname does not exist or if it contains illegal characters, you will not get an error.

?CREATE Continued

The ?CREATE packet varies, depending on whether you are creating an IPC file, a directory, or one of the other valid file types. Figure 2–13, Figure 2–15, and Figure 2–16 show the structures of the three ?CREATE packets.

IPC Entry Packet

You can create an IPC file only in the ?CREATE caller's initial working directory. When a process terminates, the operating system deletes all of its IPC files. Figure 2–13 shows the ?CREATE packet for creating an IPC file and Table 2–10 describes its contents.

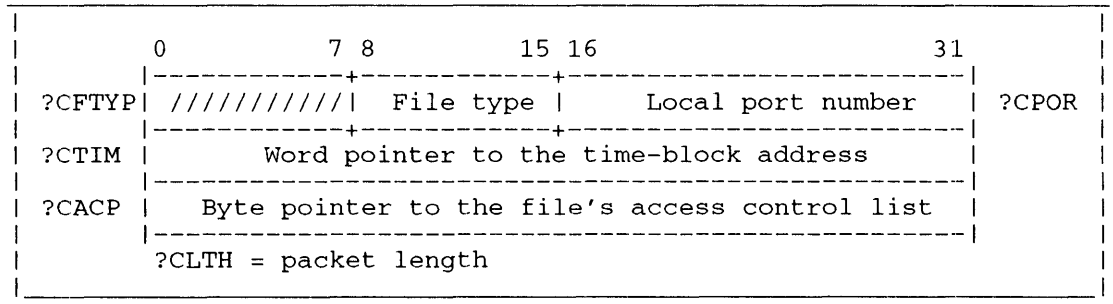


Figure 2–13. Structure of ?CREATE IPC Packet

NOTE: You can protect an IPC file by denying other users access to it. To do this, create the file either in your working directory or in one of its subdirectories. This means that other users do not have Execute access to the directory (unless you explicitly give it to them). Without Execute access to the directory, other users cannot issue ?OPEN or ?ILKUP calls against the IPC file.

Table 2–10. Contents of ?CREATE IPC Packet*

Offset	Contents
?CFTYP	Set the right byte of offset ?CFTYP to the file type for an IPC file. The valid file types are ?FGLT, ?FIPC, ?FQUE, and ?FSPR.
?CPOR	Local port number for IPC file in range from 1 through ?MXLPN. (The ring from which ?CREATE is issued determines the ring number that is associated with the public port.)
?CTIM (doubleword)	Address of new file's time block, if present. (See Figure 2-14 for the structure of the ?CREATE time block.) You must express the time-block values as double-precision integers where the high-order portion states the duration (in days) from 31 December 1967 to the date the file was created, accessed, or modified. The low-order portion states the time the file was created, accessed, or modified. You express the time value as the number of biseconds (half the number of seconds) since midnight. The time value must be less than 43,200 (decimal). DEFAULT = -1 (Set all values to current date and time).
?CACP (doubleword)	New file's access control list (ACL), which you specify elsewhere in your program. If you set ?CACP to 0, the new file will have no ACL. Use the ACL format shown in ?DACL. DEFAULT = -1 (give new file caller's default ACL).

* There is no default unless otherwise specified.

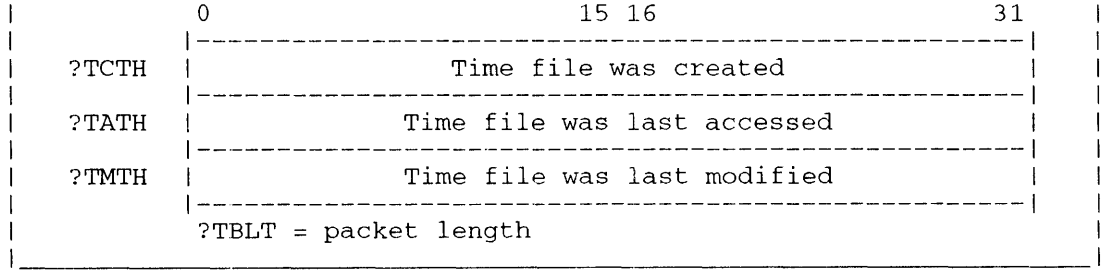


Figure 2–14. Structure of ?CREATE Time Block

Access Control Specifications

Within all three packet types, offset ?CACP points to the new file's access control list (ACL), which you must specify elsewhere in your program. If you set this parameter to –1 (the default value), the new file will have the ?CREATE caller's default ACL. If you set ?CACP to 0, the new file will have no ACL. If you specify an ACL, use the format shown in the description of ?DACL.

Directory Packet

Figure 2–15 shows the ?CREATE packet for creating a directory and Table 2–11 describes its contents. As in the other ?CREATE packets, the right byte of offset ?CFTYP specifies the new directory's file type: either ?FDIR, for a standard directory, or ?FCPD, for a control point directory.

Offsets ?CTIM and ?CACP point to the addresses of the directory's time block and ACL, respectively. The standard default for ?CTIM is –1, which sets all values in the time block to the current time. (See Figure 2–14 for the structure of the time block. Also, see "Access Control Specifications," for a description of ?CACP.)

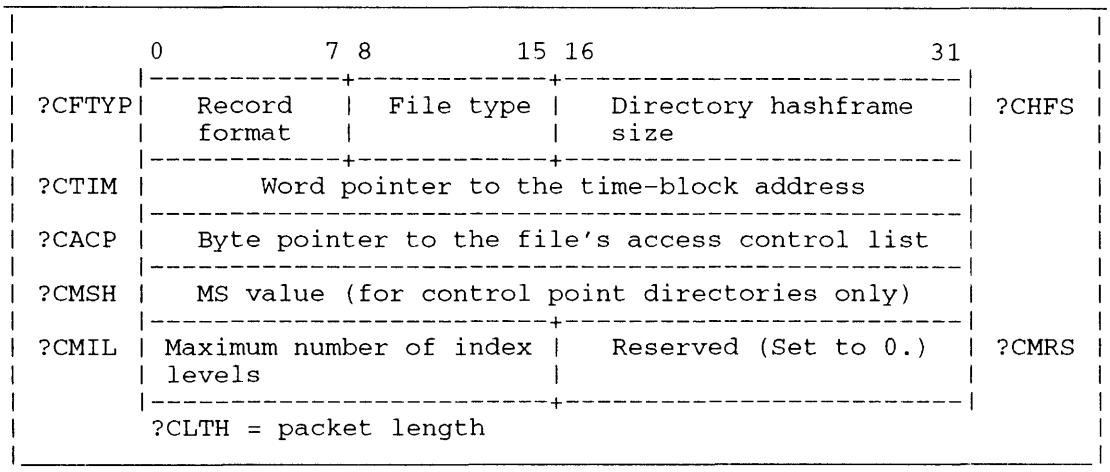


Figure 2–15. Structure of ?CREATE Directory Packet

If you are creating a control point directory (CPD), set offset ?CMSH to the maximum space value (MS) for the directory. The MS value is the maximum number of disk blocks available to the CPD and all its subordinate files, excluding files in a subordinate logical disk. (To change a CPD's MS value, use the ?CPMAX system call.)

The operating system always creates a value of 3 for the maximum number of index levels in any new directory file that it creates. The default value of –1 for ?CMIL, like all values, results in this maximum number.

If you set offset ?CMIL to 0 (no index levels), the operating system builds the directory contiguously. If you set ?CMIL to –1 (the default value) or to a value outside the legal range, the operating system creates as many index levels as necessary, to a maximum of three levels.

If you set offset ?CHFS to –1 or 0, AOS/VS assigns the default length hashframe size to the directory. For AOS/VS II, the operating system determines the hashframe size and ignores offset ?CHFS.

Table 2–11. Contents of ?CREATE Directory Packet*

Offset	Contents
?CFTYP	(Right byte.) File type of new directory: ?FDIR (for a standard directory) or ?FCPD (for a control point directory).
?CHFS	Directory hashframe size. DEFAULT = -1 (Set hashframe size to 7.)
?CTIM (doubleword)	Address of new file's time block, if present. (See Figure 2-14 for the structure of the ?CREATE time block.) DEFAULT = -1 (Set all values to current date and time).
?CACP (doubleword)	New directory's access control list (ACL), which you specify elsewhere in your program. If you set ?CACP to 0, the file will have no ACL. DEFAULT = -1 (give new file caller's default ACL).
?CMSH (doubleword)	(Control point directories only. Set to 0 for file type ?FDIR.) Maximum space value of new directory.
?CMIL	Maximum number of index levels for new directory. The operating system always creates directories with a maximum of 3 index levels. DEFAULT = -1 (set maximum number of index levels to 3).
?CMRS	Reserved (Set to 0.)

* There is no default unless otherwise specified.

Other File Types Packet

Figure 2–16 shows the structure of the packet for the remaining file types, and Table 2–12 describes its contents.

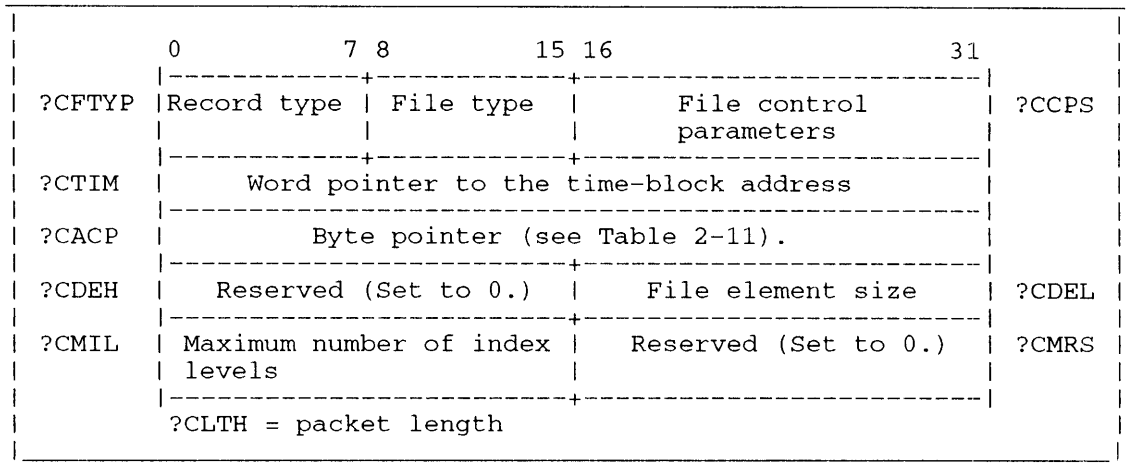


Figure 2–16. Structure of ?CREATE Packet for Other File Types

In this packet, offset ?CFTYP contains two significant parameters: record format and file type. You can specify any valid file type, except the following:

- ?FGLT, ?FIPC, ?FQUE, or ?FSPR (use the ?CREATE IPC packet).
- ?FDIR or ?FCPD (use the ?CREATE directory packet).
- Any file type that is restricted from user creation.

You can set the record format field to any one of the following masks:

?ORDS	data–sensitive
?ORDY	dynamic
?ORFX	fixed–length
?ORVR	variable–length

To place a value in the left byte of offset ?CFTYP, multiply it by 400 octal = 256 decimal. For example, suppose you want to create a user data file with dynamic records. Here's part of a parameter packet that shows one way to do this.

```
PKT:    .LOC    PKT+?CFTYP
        .WORD   (?ORDY*400)+?FUDF
```

Table 2–12. Contents of ?CREATE Packet for Other File Types*

Offset	Contents
?CFTYP	(Left byte: record type.) This can be any one of the following masks: ?ORDY (dynamic); ?ORDS (data-sensitive); ?ORFX (fixed-length); or ?ORVR (variable-length). (Right byte: file type.) This can be any file type except ?FCPD, ?FDIR, ?FGLT, ?FIPC, ?FQUE, or ?FSPR.
?CCPS	(Fixed-length records only; otherwise ignored.) Maximum record length of new file. See the description of offset ?SCPS in the explanation of system call ?FSTAT.
?CTIM (doubleword)	Address of new file's time block, if present. (See Figure 2-14 for the structure of the ?CREATE time block.) DEFAULT = -1 (set all values to current date and time).
?CACP (doubleword)	One of the following: - Byte pointer to new file's access control list (ACL). (If you set ?CACP to 0, the new file will have no ACL.) - Byte pointer to link resolution pathname if you are creating a link. DEFAULT = -1 (give new file caller's default ACL).
?CDEH	Reserved (Set to 0.)
?CDEL	Number of disk blocks per element for new file in the range from -1 through 65534. DEFAULT = -1 (set to element size selected during the system-generation procedure).
?CMIL	Maximum number of index levels for new directory in the range from 0 through 3. DEFAULT = -1 (set maximum number of index levels to 3).

* There is no default unless otherwise specified.

If you do not specify the record format, you must specify it when you open, read, or write the file.

Offset ?CCPS, the file control parameter, applies only to entries with fixed–length records. Offset ?CCPS must equal the new file's maximum record length. The operating system ignores ?CCPS if the record format is not fixed length.

Offset ?CTIM points to the address of the directory's time block. The standard default for ?CTIM is –1, which sets all values in the time block to the current time. (See Figure 2–16 for the structure of the time block.)

Offset ?CACP either points to the directory's ACL or it points to the resolution pathname of a link. (See "Access Control Specifications," in this description for more information on using offset ?CACP as a byte pointer to the ACL.)

Set offset ?CDEL to the new file's file–element size; that is, the number of disk blocks per file element. If you set this parameter to –1, the operating system assigns the default file–element size. (The default file–element size was specified during the system–generation procedure.)

AOS/VS rounds the file–element size you specify to the next multiple of the default file–element size. If the rounded file–element size is larger than 65534, the operating system assigns the file a file–element size of 65534. For example, if the system default file–element size is 4 and you specify 65533, the operating system assigns the size 65534. AOS/VS II does not round the element size.

If you set ?CMIL to 0 (no index levels), the operating system builds the directory contiguously. If you set ?CMIL to –1 (the default value) or to a value outside the legal range, the operating system creates as many index levels as necessary, to a maximum of three levels.

You can use ?CREATE to create a pipe file whose default size is 4096 bytes (2 pages), in which case offsets ?CFTYP, ?CCPS, and ?CDEL are ignored. However, ?OPEN offers you control over a pipe file's length (specify up to 20 pages in offset ?IMRS) and pending behavior. Only ?OPEN lets you use the pipe extension packet. Details appear under ?OPEN.

Sample Packet

The following sample packet shows creating an IPC file:

```
PKT:    .BLK        ?CLTH           ;Allocate enough space for packet
                                    ;(packet length = ?CLTH).

        .LOC        PKT+?CFTYP      ;File type for IPC file.
        .WORD       ?FIPC           ;Standard IPC file. (There is no
                                    ;default for ?CFTYP.)

        .LOC        PKT+?CPOR       ;Local port number.
        .WORD       5               ;Local port number is 5.

        .LOC        PKT+?CTIM       ;Address of time block.
        .DWORD      -1              ;No time block exists, so set all
                                    ;values in time block to current time
                                    ;(default = -1).

        .LOC        PKT+?CACP       ;Byte pointer to file's ACL.
        .DWORD      -1              ;Give the new file the same ACL as
                                    ;the caller.

        .LOC        PKT+?CLTH       ;End of ?CREATE IPC packet.
```

Notes

- See the description of ?OPEN in this chapter for information on the creation option.
- See the descriptions of ?OPEN, ?READ, and ?WRITE in this chapter for information on the structure of their packets.
- AOS/VS II assigns the following ?XCREATE file structure parameters to a file (or directory) that you create with ?CREATE:

  Secondary Data Element Size = (Primary) Data Element Size.

Number of Primary Elements = 1.

Index Element Size = 1.

- AOS/VS II assigns the following additional file structure parameters to a directory that you create with ?CREATE:

  Primary Data Element Size = 1.

  Maximum Index Level = Currently defined by the system at 3.

- For AOS/VS II, see the description of the ?XCREATE and ?XFSTAT system calls for the additional file structuring parameters that the ?XCREATE system call sets and the ?XFSTAT call displays.

?CRUDA *[packet address]*
error return
normal return

Input

AC0	One of the following: • Byte pointer to the pathname of the target • 0 if a packet address is supplied
AC1	Reserved (Set to 0.)
AC2	One of the following: • Reserved (Set to 0 if not supplying a packet.) • Address of the ?CRUDA packet, unless you specify the address as an argument to ?CRUDA

Output

AC0	Unchanged file
AC1	Unchanged
AC2	Undefined or address of ?CRUDA packet

Error Codes in AC0

ERFAD	File access denied
ERIFT	Illegal file type
ERNRD	Insufficient room in directory
ERUAE	User data area already exists
ERVBP	Invalid byte pointer passed as a system call argument
ERVWP	Invalid word pointer passed as a system call argument
ER_FS_DIRECTORY_NOT_AVAILABLE	Directory not available because the LDU was force released (AOS/VS II only)
ER_FS_INVALID_PATHNAME_BYTE_PTR	Invalid byte pointer to pathname
ER_FS_TLA_MODIFY_VIOLATION	Attempt to modify an AOS/VS II file with ?ODTL value supplied in ?GOPEN packet

Why Use It?

You can use UDA data to tailor a file's output specifications, provided the intended output device is a data channel line printer that is controlled by the EXEC utility. If there is no UDA defined for a file, the operating system uses the default EXEC format specifications.

Who Can Use It?

There are no special process privileges needed to issue this call. If you specified the file with a channel number, you must have Write or Owner access to the target file. If, on the other hand, you specified the file with a pathname, you must also have Execute access to the parent directory.

What It Does

A User Data Area (UDA) is often used to store a file's format descriptions, although you can use it for any other purpose. The ?CRUDA system call creates a 128–word UDA for the target file that you specify. The target file can be specified in one of two ways: either by a byte pointer to the file's pathname in AC0, or by using offset ?GCPCN in the ?CRUDA packet. The use of the packet is only necessary if you choose to specify the target file's channel number. AC0 must be set to zero to indicate the use of a packet.

Once you create a UDA with ?CRUDA, you can write to it with ?WRUDA and read from it with ?RDUDA. The UDA exists until you delete the target file. The operating system clears a UDA when it creates one.

Figure 2–17 shows the structure of the ?CRUDA packet.

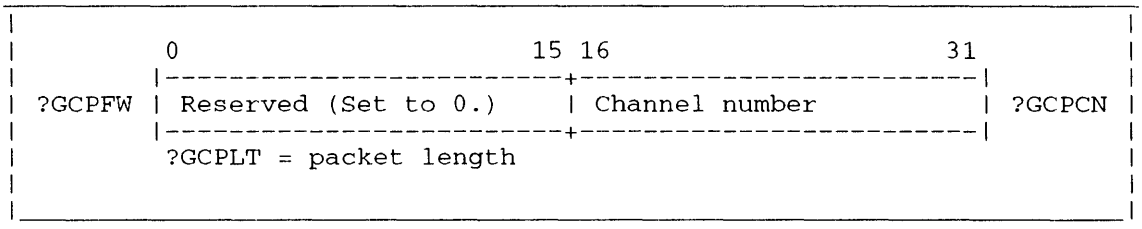


Figure 2–17. Structure of ?CRUDA Packet

Notes

- See the descriptions of ?RDUDA and ?WRUDA in this chapter.

?CTERM
error return
normal return

Input		Output
AC0	PID of the customer to terminate or a byte pointer to that customer's process name	Unchanged
AC1	If AC0 contains a byte pointer, –1	Unchanged
AC2	Reserved (Set to 0.)	Unchanged

Error Codes in AC0

ERCBK	Connection broken
ERCDE	Connection doesn't exist
ERPRH	Attempt to access process not in the hierarchy
ERPNM	Illegal process name
ERPRV	Caller not privileged for this action
ERVBP	Invalid byte pointer passed a system call argument

Why Use It?

?CTERM is one of six system calls that cause a disconnection. (The others are ?BRKFL, ?DRCON, ?RESIGN, ?RETURN, and ?TERM.) Although ?CTERM breaks the customer/server connection, it does not clear the customer/server entry from the connection table. Both processes must disconnect for this to occur. After the customer process is disconnected, the operating system terminates it.

Who Can Use It?

There are no special process privileges needed to issue this call, and there are no restrictions concerning file access. However, the calling process must be a server of the customer process that you specify.

What It Does

?CTERM terminates the customer process that you specify in AC0, and breaks all its connections with servers.

When a server issues ?CTERM, the operating system returns an obituary message from the global IPC port ?SPTM. The obituary message notifies the server of the customer's demise.

To receive the obituary message, the server must issue a global ?IREC against port ?SPTM. The operating system passes the obituary message (termination code ?TBCX) to offset ?IUFL in the server's ?IREC header. (If the server is the customer's father, it must issue two ?IREC system calls: one for the obituary message, and one for the termination code.)

Notes

- See the description of ?IREC in this chapter.
- See the descriptions of ?BRKFL, ?CON, ?DCON, ?RESIGN, ?RETURN, and ?TERM (the disconnect system calls) in this chapter.

?CTOD
error return
normal return

Input

AC0	Scalar time you want to convert expressed as the number of biseconds (seconds/2) since midnight in Bits 16 through 31
AC1	Reserved (Set to 0.)
AC2	Reserved (Set to 0.)

Output

AC0	Seconds from 0 through 59
AC1	Minutes from 0 through 59
AC2	Hour from 0 (midnight) through 23 (11 p.m.). (The result is expressed in octal.)

Error Codes in AC0

No error codes are currently defined.

Why Use It?

The system clock maintains the current time in the standard hours, minutes, and seconds, where the value for the current hour can range from 0 (midnight) through 23 (11 p.m.). Both the ?FSTAT packet and the ?CREATE time block express the time in scalar notation. Thus, you can use ?CTOD to convert the values returned by ?FSTAT or ?CREATE.

Who Can Use It?

There are no special process privileges needed to issue this call, and there are no restrictions concerning file access.

What It Does

?CTOD converts a scalar time value, specified in AC0, to its equivalent in hours, minutes, and seconds. The scalar value for a particular time is the number of biseconds (half the number of seconds) that have elapsed since midnight.

?CTYPE
error return
normal return

Input

AC0 One of the following:

- Byte pointer to the name of the target process
- PID of the target process
- –1 to change the process type of the calling process

AC1 One of the following:

- –1 if AC0 contains a byte pointer
- 0 if AC0 contains a PID

AC1 is ignored if AC0 contains –1

AC2 New process type. The process type parameters are:

- 0 for swappable processes
- ?PFRP for pre–emptible processes
- ?PFRS for resident processes

Output

AC0 Unchanged

AC1 Unchanged

AC2 Unchanged

Error Codes in AC0

ERMPR	System call parameter address error
ERPNM	Illegal process name format
ERPRH	Attempt to access a process not in hierarchy
ERPRP	Illegal process priority
ERPTY	Illegal process type
ERVBP	Invalid byte pointer passed as a system call argument
ERWSF	Working set not swappable (The working set has too many wired pages and, therefore, this process cannot become nonresident.)

Why Use It?

?CTYPE lets you change a process's type during program execution, thereby possibly changing the process's priority for a system resource. You can use ?CTYPE to favor the target process or to favor another process over the target process.

?CTYPE Continued

Who Can Use It?

To issue ?CTYPE, the calling process must have privilege ?PVTY assigned in its ?PROC packet. The calling process can use ?CTYPE to change the type of any subordinate process. However, if the calling process is in Superprocess mode, it can change the type of any process. There are no restrictions concerning file access.

What It Does

?CTYPE changes the process type of the process that you specify in AC0 (the calling process itself, one of its sons, or any other process).

NOTE: When a 16–bit process becomes resident, the operating system implicitly wires its Ring 7 pages, and then unwires them if the process changes to another process type.

Notes

- See the descriptions of ?PROC and ?WIRE in this chapter.

?DACL Sets, clears, or examines a default access control list.

?DACL
error return
normal return

Input

AC0	One of the following: • –1 to set a new default ACL to the string that the byte pointer in AC1 specifies • 0 to return the current default ACL string, and the status (DEFACL ON or OFF) in AC0 • 1 to turn off the default ACL mode
AC1	Byte pointer to a 128–word buffer if AC0 contains –1; if AC0 = 0, then AC1 contains a byte pointer to a 128–word receive buffer
AC2	Reserved (Set to 0.)

Output

AC0	The default ACL mode (onor off), if AC0 = 0 on input; otherwise, unchanged
AC1	Unchanged
AC2	Undefined

Error Codes in AC0

ERACL	Illegal ACL
ERPRE	Invalid system call parameter
ERVBP	Invalid byte pointer passed as a system call argument
ER_FS_INVALID_DEFACL_OPTION	Invalid ?DACL option specified

Why Use It?

You can use ?DACL to assign your process and one or more other processes (which you specify by their usernames) access to all files that your process will create. A default ACL that you define this way is valid until the ?DACL caller terminates or issues another ?DACL to redefine the default ACL.

Who Can Use It?

There are no special process privileges needed to issue this call, and there are no restrictions concerning file access.

?DACL Continued

What It Does

Depending on the input value of AC0, ?DACL lets you define a default ACL for the duration of the calling process, clear the default ACL, or obtain the current default ACL. To supply an ACL string, use the following format:

username<0>accesstype[...]<0>

where

accesstype is one of the following:

?FACA	Append access
?FACE	Execute access
?FACR	Read access
?FACW	Write access
?FACO	Owner access

Two examples of supplied ACL strings are

JEFF<0><?FACO+?FACW+?FACA+?FACR+?FACE><0>

and

JEFF<0><?FACO+?FACW+?FACA+?FACR+?FACE>LISA<0><?FACR+?FACE><0>

If you do not supply an ACL string, the operating system sets the default ACL to full access for the calling process and all processes that have the calling process's username:

caller's_username<0><?FACO+?FACW+?FACA+?FACR+?FACE><0>

To examine the current ACL, load AC0 with 0. The operating system then returns the current default ACL to the buffer you specify in AC1.

Note that the default ACL keys on specific usernames, that is, any usernames that the calling process cites in the ACL string, rather than keying on specific filenames.

?DADID
error return
normal return

Input

AC0	One of the following: • PID of the target process • –1 to obtain the PID of the caller's father process
AC1	Reserved (Set to 0.)
AC2	Reserved (Set to 0.)

Output

AC0	Unchanged
AC1	PID of the target process's father
AC2	Undefined

Error Codes in AC0

ERPRH	Attempt to access process not in hierarchy
ERPOR	PID is out of range for this process

Why Use It?

A number of system calls require a PID as an input parameter. ?DADID provides you with a simple way to identify a PID, which you can then use as an input parameter for another system call.

Who Can Use It?

There are no special process privileges needed to issue this call, and there are no restrictions concerning file access.

What It Does

?DADID returns the PID of a target process's father. The target process, which you specify in AC0, can be either the caller or some other process in the process tree.

?DCON
error return
normal return

Input		Output	
AC0	Reserved (Set to 0.)	AC0	Undefined
AC1	PID of the target process in the connection	AC1	Unchanged
AC2	Reserved (Set to 0.)	AC2	Undefined

Error Codes in AC0

ERCBK	Connection has been broken
ERCDE	Connection doesn't exist (Two likely errors: the target process does not exist; or the target process is not connected with the caller.)

Why Use It?

?DCON breaks the Ring 7 connection from either the caller's or the server's end. (To break a connection to a specific ring other than Ring 7, use ?DRCON.)

Who Can Use It?

There are no special process privileges needed to issue this call, and there are no restrictions concerning file access.

What It Does

?DCON breaks the Ring 7 connection between the calling process and the process (customer or server) that you specify in AC1. Both servers and customers can issue ?DCON.

Note that a single ?DCON does not clear the customer/server connection from the connection table. For this to occur, both processes must break the connection. If both processes do not break the connection, the operating system reserves the PID number and does not reuse it until the connection is cleared.

Notes

- See the description of ?CON (the connection system call) and the descriptions of ?BRKFL, ?CTERM, ?DRCON, ?RESIGN, ?RETURN, and ?TERM (which disconnect processes) in this chapter. System calls ?BRKFL, ?CTERM, ?RETURN, and ?TERM are also termination calls.
- ?DCON is provided primarily to provide compatibility with 16–bit programs converted from AOS. ?DRCON is the preferred system call.

?DDIS — Disables access to all devices.

?DDIS
error return
normal return

Input

None

Output

None

Error Codes in AC0

ERPRV Caller not privileged for this action

Why Use It?

?DDIS is the opposite of ?DEBL, which enables access to all user–defined and system–defined devices. You can issue ?DDIS to disable device access after completing a task I/O routine that began with ?DEBL.

Who Can Use It?

The calling process must have privilege ?PVDV to issue this call. There are no restrictions concerning file access.

What It Does

?DDIS disables a process's access to all system and user devices.

?DDIS does not re–enable LEF mode. Therefore, to re–enable LEF mode, you must issue ?LEFE (enable LEF mode).

To re–enable device I/O after a ?DDIS, issue ?DEBL.

?DDIS affects only the segment that issued the call.

?DDIS turns off the "issue I/O instructions" privilege for a segment. See the *ECLIPSE® MV/Family (32–Bit) Systems Principles of Operation* manual.

?DEASSIGN
error return
normal return

Input

AC0	Byte pointer to the name of the device that you want to deassign
AC1	Reserved (Set to 0.)
AC2	Reserved (Set to 0.)

Output

AC0	Unchanged
AC1	Undefined
AC2	Undefined

Error Codes in AC0

ERARC	Attempt to release terminal device
ERARU	Attempt to release an unassigned device
ERIDT	Illegal device name type
ERVBP	Invalid byte pointer passed as a system call argument

Why Use It?

If you used ?ASSIGN to assign a character device to a process, you must use ?DEASSIGN to cancel the assignment. The operating system also allows implicit device assignments and cancellations with ?OPEN and ?CLOSE.

Who Can Use It?

There are no special process privileges needed to issue this call, and there are no restrictions concerning file access.

What It Does

?DEASSIGN breaks the explicit assignment you established between your process and a device with ?ASSIGN. Note that you cannot use ?DEASSIGN to break the explicit assignent if the process's console file is your process's console file or if you have opened the console file.

Notes

- See the description of ?ASSIGN in this chapter.
- See the descriptions of ?OPEN and ?CLOSE in this chapter for information on implicit device assignments and cancellations.

?DEBL — Enables access to all devices.

?DEBL
error return
normal return

Input

None

Output

None

Error Codes in AC0

ERPRV Caller not privileged for this action

Why Use It?

You would probably issue ?DEBL before entering a task–level I/O sequence. ?DEBL gives the calling process global access to all devices, including system–defined devices.

Who Can Use It?

The calling process must have privilege ?PVDV to issue this call. There are no restrictions concerning file access.

What It Does

?DEBL turns off I/O protection and disables LEF mode. This allows a program to issue instructions to *any* I/O device without operating system intervention.

Because ?DEBL turns off I/O protection and disables LEF mode, tasks within the calling process can successfully issue I/O instructions against devices.

After the operating system executes ?DEBL, it interprets all subsequent LEF instructions as I/O instructions, until you issue ?LEFE to restore LEF mode.

?DEBL affects only the segment that issued the call.

?DEBUG — Calls the Debugger utility.

?DEBUG
error return
normal return

Input

None

Output

None

Error Codes in AC0

No error codes are currently defined.

Why Use It?

If you pass control to the user debugger directly from a program, you can examine memory locations and/or execution paths while the program runs. (Another way to call the Debugger utility is to choose the ?PFDB option in offset ?PFLG of the ?PROC packet.)

Who Can Use It?

There are no special process privileges needed to issue this call, and there are no restrictions concerning file access.

What It Does

?DEBUG passes control to the operating system's Debugger utility while your program is executing. To return control to the program (that is, to the ?DEBUG normal return), issue the Debugger command $P (ESC P).

Notes

- See the description of ?PROC in this chapter.

?DELAY — Suspends a 16–bit task for a specified interval (16–bit processes only).

?DELAY
error return
normal return

Input

AC0 and AC1
: Delay interval (in milliseconds) in Bits 16 through 31 of each accumulator

AC2 Reserved (Set to 0.)

Output

AC0 and AC1
: Modified (The OS uses these accumulators to maintain delays.)

AC2 Undefined

Error Codes in AC0

No error codes are currently defined.

Why Use It?

?DELAY allows a task to wait until a specified amount of time has passed.

Who Can Use It?

There are no special process privileges needed to issue this call, and there are no restrictions concerning file access.

What It Does

?DELAY, which is the 16–bit counterpart of ?WDELAY, suspends the calling task for the number of milliseconds that you specify in AC0 and AC1. You must express the delay interval as a double–precision integer in Bits 16 through 31 of AC0 and AC1.

If the number of milliseconds that you specify is not a multiple of the real–time clock frequency, the operating system rounds it to the next highest frequency multiple. For example, if the clock frequency is 10 hertz (one period = 100 milliseconds) and you specify a delay of 120 milliseconds, the operating system rounds the delay interval to 200 milliseconds. To obtain the real–time clock frequency, issue ?GHRZ.

Notes

- See the descriptions of ?WDELAY and ?GHRZ in this chapter.

?DELETE *[packet address]*
error return
normal return

Input

AC0 One of the following:

- Byte pointer to the pathname of the target file
- 0 if a packet address is supplied

AC1 Reserved (Set to 0.)

AC2 One of the following:

- Reserved (Set to 0 if not supplying a packet.)
- Address of the ?DELETE packet, unless you specify the address as an argument to ?DELETE

Output

AC0 Unchanged

AC1 Undefined

AC2 Undefined or address of ?DELETE packet

Error Codes in AC0

ERDID Directory delete error (You tried to delete a directory that contains one or more subordinate directories.)

ERDIU Directory in use — cannot delete (You tried to delete a working directory or a directory cited in a search list.)

ERFDE File does not exist

ERPRM Cannot delete permanent file (You tried to delete a file or directory with the Permanence attribute.)

ERVBP Invalid byte pointer passed as a system call argument

ERWAD Write access denied

ERCDR Can't delete the root directory

ER_FS_DIRECTORY_NOT_AVAILABLE
Directory not available because the LDU was force released (AOS/VS II only)

ER_FS_TLA_MODIFY_VIOLATION
Attempt to modify an AOS/VS II file with ?ODTL value supplied in ?GOPEN packet

Why Use It?

By deleting a file or directory, you can reclaim disk space or, in the case of an IPC file, renumber the port.

When you issue ?DELETE against an open file, the operating system deletes the filename immediately, and then waits until all users have closed the file before it actually deletes the file. ?DELETE also deletes any user data area (UDA) associated with the file. If the input pathname does not begin with a prefix character (:, ^, or =), ?DELETE assumes a prefix of =. This means that ?DELETE will not scan the caller's search list to find the target file.

086-000195 updates
093-000542

Who Can Use It?

If you specified the file with a channel number, the calling process must have Write access to the target entry's directory or Owner access to the target entry. If, on the other hand, you specified the file with a pathname, you must also have Execute access to the parent directory.

What It Does

?DELETE deletes a file or directory. The target entry can be specified in one of two ways: either by a byte pointer to the entry's pathname in AC0, or by using offset ?GCPCN in the ?DELETE packet. The use of a packet is only necessary if you choose to specify the target entry's channel number. AC0 must be set to zero to indicate the use of a packet.

If a byte pointer to the entry's pathname is specified in AC0 and if the last file in the pathname is a link entry, then the operating system will delete the link entry without resolving it. Therefore, the directories or files to which the link refers remain intact.

As the error codes indicate, you cannot delete a process's working directory or any directory cited in a search list.

Figure 2–18 shows the structure of the ?DELETE packet.

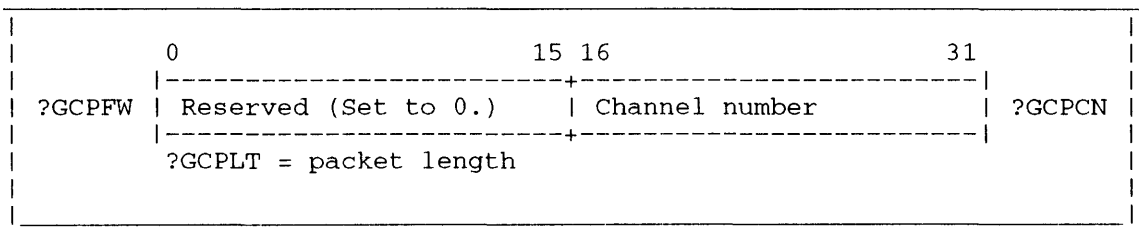


Figure 2–18. Structure of ?DELETE Packet

Notes

- See the description of ?CREATE in this chapter for information on creating files and directories.

?DFRSCH — Disables task rescheduling and indicates prior state of rescheduling.

?DFRSCH
error return
normal return

Input

None

Output

AC0 ?DSCH if task rescheduling is disabled when you issue ?DFRSCH

AC1 Undefined

AC2 Undefined

Error Codes in AC0

No error codes are currently defined.

Why Use It?

You can use ?DFRSCH to lock all other tasks out of a critical region and still retain a copy of the previous state of multitask scheduling in memory. Internally called subroutines that need to disable scheduling can use ?DFRSCH, rather than ?DRSCH, to restore previous multitask scheduling environments when the operating system exits from the subroutine.

As with ?DRSCH, you should be very careful when you use ?DFRSCH, because ?DFRSCH disrupts multitask activity for the entire process.

Who Can Use It?

There are no special process privileges needed to issue this call, and there are no restrictions concerning file access.

What It Does

?DFRSCH blocks all multitask scheduling for the current process until you explicitly re–enable scheduling with ?ERSCH or until you terminate the calling task with ?KILL. The calling task retains CPU control while scheduling is disabled, even if other tasks are ready to run.

When ?DFRSCH succeeds, it returns a flag in AC0 that indicates whether multitask scheduling was enabled before ?DFRSCH took action. A value of ?DSCH in AC0 indicates that multitask scheduling was disabled before you issued ?DFRSCH.

Notes

- See the descriptions of ?ERSCH and ?DRSCH in this chapter.
- Never issue a system call that, after you have disabled task scheduling, could pend on an event from another task. Doing this could cause your process to hang.

?DIR
error return
normal return

Input

AC0 One of the following:

- Byte pointer to the pathname of the new working directory (that is, the destination directory)
- 0 to specify the initial working directory

AC1 Reserved (Set to 0.)

AC2 Reserved (Set to 0.)

Output

AC0 Unchanged

AC1 Undefined

AC2 Undefined

Error Codes in AC0

ERDAD Directory access denied
ERVBP Invalid byte pointer passed as a system call argument
ER_FS_DIRECTORY_NOT_AVAILABLE
Directory not available because the LDU was force released (AOS/VS II only)
ER_FS_INVALID_PATHNAME_BYTE_PTR
Invalid byte pointer to pathname

Why Use It?

A process's working directory is its starting point for file access. You can change your working directory to access files that are not contained in your current working directory. ?DIR also lets you return to your initial directory after you work elsewhere.

Who Can Use It?

There are no special process privileges needed to issue this call. The calling process must have Execute access to a directory to use it as a working directory. The calling process must also have Execute access to the target directory's parent directory.

What It Does

?DIR changes the caller's working directory to the directory that you specify in AC0 or to the caller's initial working directory (if AC0 contains 0).

?DQTSK — Removes from the queue one or more previously queued tasks.

?DQTSK *[task definition packet address]*
error return
normal return

Input

AC0	Reserved (Set to 0.)
AC1	Reserved (Set to 0.)
AC2	Address of the task definition packet to remove from the initiation queue, unless you specify the address as an argument to ?DQTSK

Output

AC0	Undefined
AC1	Undefined
AC2	Address of the task definition packet

Error Codes in AC0

ERVWP	Invalid word pointer passed as a system call argument
ERQTS	Error in qtask request (There is no match on the task definition packet.)

Why Use It?

?DQTSK can be useful for error handling, because it lets you remove one or more tasks previously placed on the execution queue with ?IQTSK and ?TASK. ?DQTSK has no effect if the task has already begun to execute.

Who Can Use It?

There are no special process privileges needed to issue this call, and there are no restrictions concerning file access.

What It Does

?DQTSK removes a specific task or tasks (as defined in the task definition packet) from the initiation queue you previously established with ?IQTSK.

?DQTSK takes the same task definition packet that you passed to ?TASK when you initiated the target task or tasks. (See Figure 2–19.) You can specify the packet address as an argument to ?DQTSK, or you can load the address into AC2 before you issue ?DQTSK. You should not alter the task definition packet before you issue ?DQTSK.

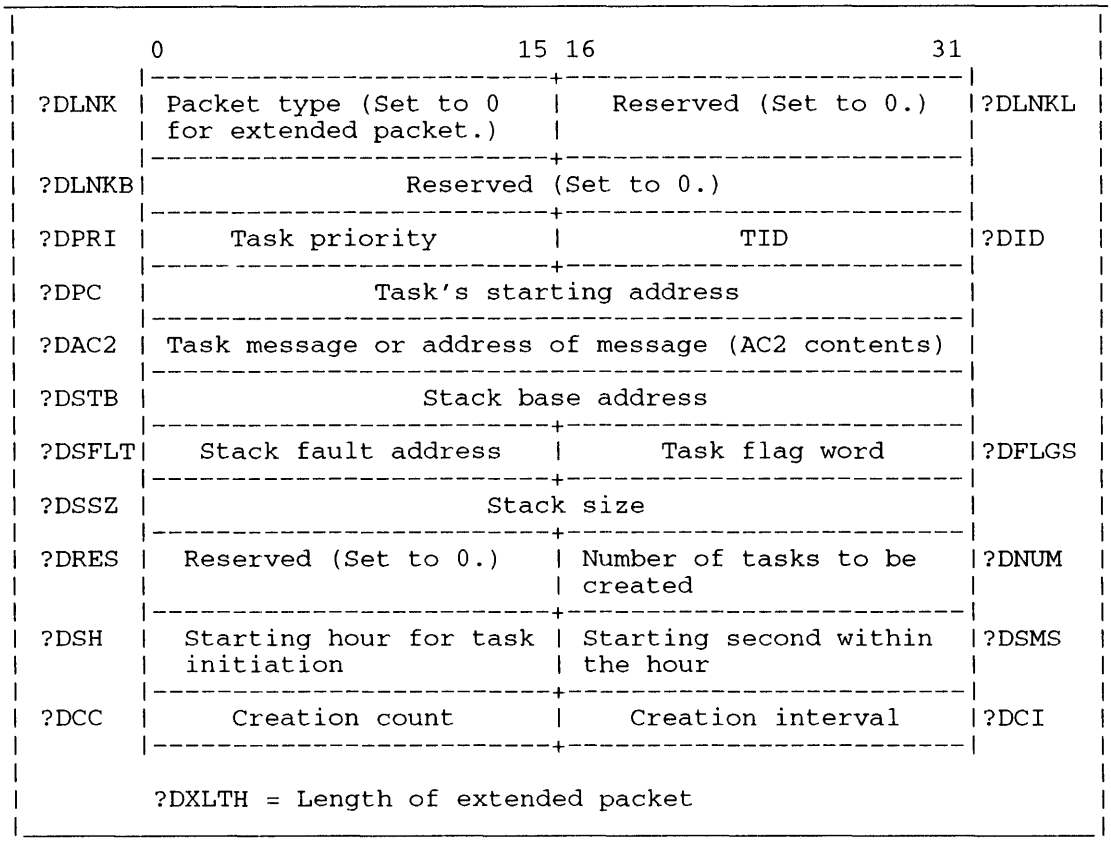


Figure 2–19. Extended Task Definition Packet

If you issue ?DQTSK against an active task, it remains active. The operating system simply pulls the task definition packet from the queue, if it is on the queue. If you want to requeue the task, you must issue ?TASK again.

Notes

- See the descriptions of ?IQTSK and ?TASK in this chapter.

?DRCON
error return
normal return

Input

AC0	Reserved (Set to 0.)
AC1	PID of the target process in the connection
AC2	Contains the following: • Bits 0 through 28 are reserved (Set to 0.) • Bits 29 through 31 contain the ring of the target in the specified process

Output

AC0	Unchanged
AC1	Unchanged
AC2	Unchanged

Error Codes in AC0

ERCBK	Connection has been broken
ERCDE	Connection doesn't exist (Two likely errors: the target process does not exist; or the target process is not connected with the caller.)
ERRNI	Invalid ring number

Why Use It?

You can use ?DRCON to break a connection to a specific ring from either the customer's or the server's end.

Who Can Use It?

There are no special process privileges needed to issue this call, and there are no restrictions concerning file access.

What It Does

?DRCON is similar to ?DCON in that it breaks the connection between the calling process and the process (customer or server) that you specify in AC1. However, unlike ?DCON, ?DRCON lets you specify the ring field in AC2. Both servers and customers can issue ?DRCON.

Note that a single ?DRCON does not clear the customer/server connection from the connection table. For this to occur, both processes must break the connection. If both processes do not break the connection, the operating system reserves the PID number and does not reuse it until the connection is cleared.

Notes

- See the description of ?CON and ?DCON in this chapter.

?DRSCH
error return
normal return

Input

None

Output

AC0 Unchanged

AC1 Unchanged

AC2 Unchanged

Error Codes in AC0

No error codes are currently defined.

Why Use It?

You can use ?DRSCH to lock all tasks, except the calling task, out of a critical region. In contrast, ?XMT, ?REC, ?IDSUS, ?PRSUS, ?IDRDY, and ?PRRDY give you finer control over resource allocation. Use ?DRSCH with discretion, if at all; ?DRSCH disrupts multitasking activity for the entire process.

Who Can Use It?

There are no special process privileges needed to issue this call, and there are no restrictions concerning file access.

What It Does

?DRSCH suppresses all multitask scheduling for the current process until you explicitly re–enable scheduling with ?ERSCH or terminate the calling task with ?KILL. The calling task retains CPU control while scheduling is disabled, even if other tasks are ready to run. (Note that if the calling task becomes suspended, so will the entire process.)

Notes

- See the descriptions of ?DFRSCH, ?ERSCH, and ?KILL in this chapter.
- Never issue a system call that, after you have disabled task scheduling, could pend on an event from another task. Doing this could cause your process to hang.

AOS/VS

?ENBRK
error return
normal return

Input

AC0 One of the following:

- Byte pointer to the name of the target process
- PID of the target process
- –1 to enable a break file for the calling process

AC1 One of the following:

- –1 if AC0 is a byte pointer
- 0 if AC0 is a PID
- Ignored if AC0 contains –1

AC2 One of the following:

- Address of the ?ENBRK packet
- –1 to cause a default dump of Ring 7 unshared and shared

Output

AC0 Unchanged or error code

AC1 Unchanged

AC2 Unchanged

Error Codes in AC0

ERNBK	No break file enabled for this ring
ERPRE	Invalid argument passed as system call parameter
ERPRV	Caller not privileged for this action
ERRNI	Invalid ring
ERVBP	Invalid byte pointer passed as system call parameter
ERVWP	Invalid address passed as system call parameter

Why Use It?

?ENBRK lets you get a memory image of any user ring of a process if the process traps, issues a Ctrl–C Ctrl–B, or is the target of a ?TERM system call. This memory image is useful in debugging programs.

NOTE: The operating system always makes a break file for a process that issues a Ctrl–C Ctrl–E or that is the target of a ?TERM/?BRKFL system call, regardless of whether or not the process issues ?ENBRK.

Who Can Use It?

There are no special process privileges needed to issue this call against the calling process. The target process must have Write access to the directory for which the break file is intended at the time the process terminates. The target process must also have Read access to the ring .PR programs that will be dumped. If the target process is denied directory access or if the control point directory maximum size is exceeded, then the operating system does not create the break file.

You can issue ?ENBRK against the calling process for all rings greater than or equal to the ring of the task that is making the call. No other privileges are necessary. Also, you can issue ?ENBRK against any ring or process if the calling process has Superprocess set on or is a server with a valid connection to the target.

When a server issues ?ENBRK against a customer, the ring that you specify as the one you want to dump must be greater than or equal to the ring of the connection. Note that the hierarchy structure is not sufficient to issue ?ENBRK. This means that a process cannot issue ?ENBRK against its son unless it has Superprocess set on, or unless it has a valid connection to its son.

What It Does

?ENBRK enables the operating system to take a break file for the specified ring if the target of the ?ENBRK traps, does a Ctrl–C Ctrl–B, or is the target of a ?TERM system call. ?ENBRK lets you specify a ring to dump, a name of a file to dump to, or a directory in which to create a dump file with a default filename.

An ?ENBRK overrides all previous ?ENBRK system calls to the specified ring of the process.

To stop creating a break file when a process terminates abnormally, reissue ?ENBRK with the offset ?ENBFL containing only a ring number, for example one of the following: ?ENR4, ?ENR5, ?ENR6, or ?ENR7. You must issue a separate ?ENBRK call for each ring in which you want to stop break files being created.

If AC2 does not contain –1 (–1 specifies the default dump of memory), you must supply a packet. Figure 2–20 shows the structure of the packet and Table 2–13 describes its contents.

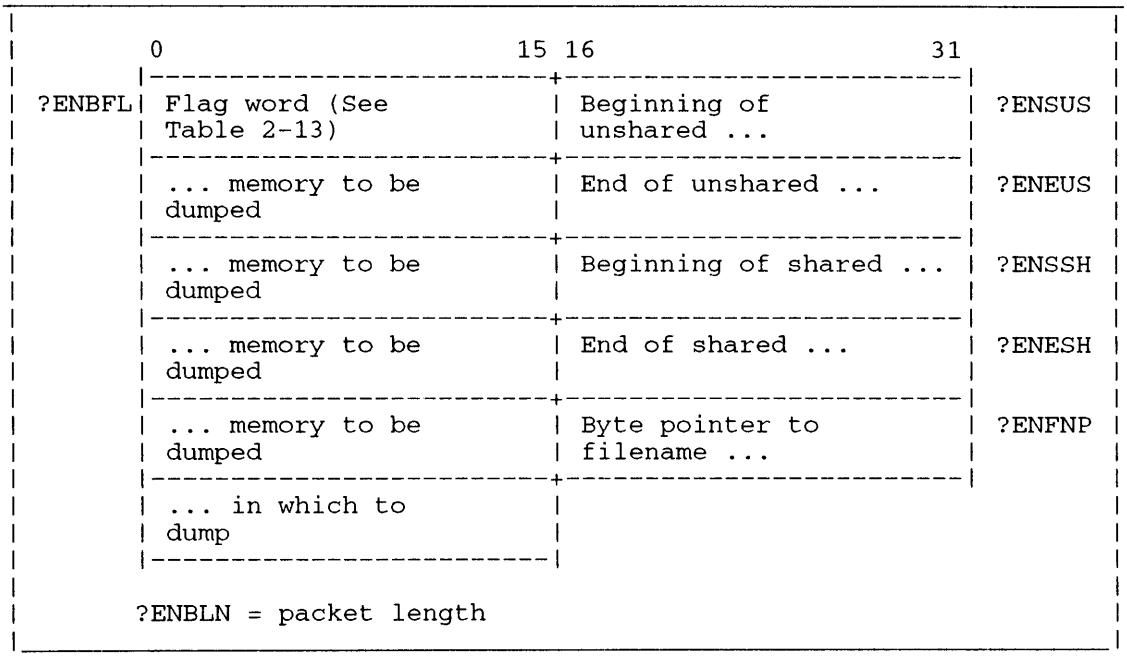


Figure 2–20. Structure of ?ENBRK Packet

Table 2–13. Contents of ?ENBRK Packet*

Offset	Contents
?ENBFL	Flag word ?ENUS---Dump unshared memory. ?ENSH---Dump shared memory. ?ENPRE--Dump preamble information. (You will get a preamble if you specify either ?ENUS or ?ENSH.) ?ENCST--Return current status. ?ENDIR--Indicate that ?ENFNP is a byte pointer to a directory. (If you do not set this flag, ?ENFNP must be a byte pointer to a pathname or -1.)
?ENR4 ?ENR5 ?ENR6 ?ENR7	?ENR4 = 4, and enables a break file for Ring 4. ?ENR5 = 5, and enables a break file for Ring 5. ?ENR6 = 6, and enables a break file for Ring 6. ?ENR7 = 7, and enables a break file for Ring 7. Used alone, each offset stops break files being created in the specified ring.
?ENSUS (double-word)	Beginning of unshared memory to be dumped. DEFAULT= -1 (to start dumping at beginning of unshared area).

* There is no default unless otherwise specified. (continued)

Table 2–13. Contents of ?ENBRK Packet*

Offset	Contents
?ENEUS (double-word)	End of unshared memory to be dumped. DEFAULT= -1 (to dump to end of unshared area).
?ENSSH (double-word)	Beginning of unshared memory to be dumped. DEFAULT= -1 (to start dumping at beginning of shared area).
?ENESH (double-word)	End of shared memory to be dumped. DEFAULT= -1 (to dump to end of shared area).
?ENFNP (double-word)	Byte pointer to filename to dump to. If ?ENDIR is set, ?ENFNP contains a byte pointer to the directory in which you want to create the default dump file. DEFAULT= -1 (to create default dump file in working directory). NOTE: A default filename takes the form ?PID.TIME.RING.BRK where PID is the 5-digit PID of the process being dumped. TIME is the current time in the form hh_mm_ss. RING is the ring number of the ring being dumped. An example of such a filename is ?00071.07_31_52.7.BRK

* There is no default unless otherwise specified. (concluded)

?ENQUE
error return
normal return

Input

AC0	Byte pointer to pathname of the spooler file or the IPC file
AC1	Byte pointer to the message
AC2	Reserved (Set to 0.)

Output

AC0	Unchanged
AC1	Unchanged
AC2	Undefined

Error Codes in AC0

ERIDT	Illegal device name type
ERPRE	Invalid system call parameter
ERVBP	Invalid byte pointer passed as a system call argument

Why Use It?

?ENQUE lets you send free–form messages to an IPC file or to a spooler file.

Who Can Use It?

There are no special process privileges needed to issue this call, and there are no restrictions concerning file access.

What It Does

?ENQUE uses the IPC facility to send a message on the global port that the IPC file or the spooler file specifies.

You must terminate the message string with a null. The string can consist of as many as 512 characters, including the terminating null.

Notes

- See the descriptions of the system calls ?ILKUP and ?ISEND for more information about global ports.

?ERMSG
error return
normal return

Input

AC0	Error code associated with the error message text string
AC1	Contains the following: • Bits 16 through 23 contain the byte length of the message buffer • Bits 24 through 31 contain the channel number assigned to the error message file (Use 377 for the system default ERMES file.)
AC2	Byte pointer to the message buffer

Output

AC0	Byte length of the error message
AC1	If the system default ERMES file is closed and Bits 24–31 contain 377, AC1 contains the channel number of the system default ERMES file. If the system default ERMES file is closed and Bits 24–31 do not contain 377, AC1 contains –1. If the system default ERMES file is open, AC1 contains the channel number of the system default ERMES file.
AC2	Unchanged

Error Codes in AC0

ERTXT	Message text longer than specified (Your message buffer is too small.)
ERVBP	Invalid byte pointer passed as a system call argument

Why Use It?

You can use ?ERMSG to get the text string associated with a particular error code.

Who Can Use It?

There are no special process privileges needed to issue this call. You must have Read access to the error message file.

What It Does

?ERMSG returns the text string associated with an error code (a 32–bit unsigned value) in either the system error message file, ERMES, or in your own error message file.

Before you issue ?ERMSG, perform the following steps:

1. Load AC0 with the error code associated with the message you want to read.

2. Load Bits 16 through 23 of AC1 with the byte size of the buffer you have set aside in your logical address space.

3. Load Bits 24 through 31 with the channel number of the error message file. (If you are reading the default ERMES file, use 377 as the channel number; the operating system returns the actual channel number. If you are reading another error message file, you must open the file with ?OPEN first. Then, you can use the channel number returned in the ?OPEN packet for ?ERMSG.)

If you pass an unknown error code to AC0, ?ERMSG takes the normal return, but the operating system passes the message "UNKNOWN ERROR CODE n" to the message buffer, where n is the ASCII equivalent of the error code you specified in AC0.

Figure 2–21 shows the structure of the error codes in ERMES. Use this structure for your own error message file if you want to use ?ERMSG to read it.

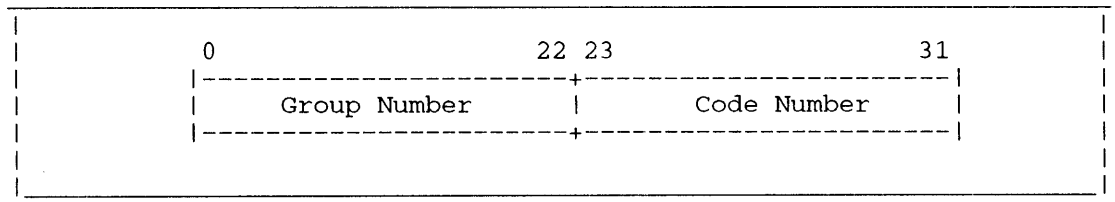


Figure 2–21. Error Code Structure in ERMES File

As Figure 2–21 shows, the error code entries in ERMES consist of two fields: group number, and code number. All error codes are categorized into groups ranging from 0 through octal 17777. Although groups 0 through octal 77 and octal 200 through octal 7777 are reserved for system definition, you can define error codes for groups octal 100 through octal 177 and octal 10000 through octal 17777.

You can change the error codes and text strings that your program reports. One important step is to edit file LINK_ERMES.CLI. See the documentation in file ERMES.SR, whose full pathname is usually :UTIL:ERMES.SR.

Two MASM macros, GRP and CODE, let you add new code groups and/or single codes to the error message file. The GRP macro defines the code group and the number of codes within that code group. The syntax of this macro is

GRP m n

where

m is the code group number.

n is the maximum number of error codes within the code group.

For example,

```
GRP 100 200
```

defines code group 100, which can have up to 200 individual error codes.

The CODE macro inserts error codes and their associated text strings into an error code group. The syntax of the CODE macro is

CODE code–mnemonic .TXT *message–string*

For example,

```
CODE   ERFTM .TXT   *FILE/TAPE DENSITY MISMATCH*
```

associates the message string FILE/TAPE DENSITY MISMATCH with the error code ERFTM.

?ERSCH Enables multitask scheduling for the calling process.

?ERSCH
error return
normal return

Input

None

Output

None

Error Codes in AC0

No error codes are currently defined.

Why Use It?

If a task disables scheduling with ?DRSCH, it has only two options to re–enable scheduling: the task can either ?KILL itself or it can issue ?ERSCH. Thus, you can use ?ERSCH to re–enable scheduling and maintain the calling task.

Who Can Use It?

There are no special process privileges needed to issue this call, and there are no restrictions concerning file access.

What It Does

?ERSCH re–enables multitask scheduling for the calling process. After the operating system executes ?ERSCH, it passes control to the normal return, and resumes its regular scheduling activities. The calling task may or may not resume execution, however, depending on its priority. If scheduling has already been enabled, it remains enabled, and ?ERSCH takes the normal return.

Notes

- See the descriptions of ?DRSCH, ?DFRSCH, and ?KILL in this chapter.

?ESFF
error return
normal return

Input

AC0	Reserved (Set to 0.)	
AC1	Channel number of the file to be flushed	
AC2	Reserved (Set to 0.)	

Output

AC0	Unchanged
AC1	Unchanged
AC2	Undefined

Error Codes in AC0

ERICN Illegal channel
ERSIM Simultaneous I/O
ERIFT Illegal file type
ER_FS_DIRECTORY_NOT_AVAILABLE
Directory not available because the LDU was force released (AOS/VS II only)
ER_FS_TLA_MODIFY_VIOLATION
Attempt to modify an AOS/VS II file with ?ODTL value supplied in ?GOPEN packet

Why Use It?

The ?ESFF call ensures that if the system fails, you do not lose any changes to shared files up to the time you issued ?ESFF. When you issue ?ESFF, the operating system saves all your changes in the disk file. This means that in an emergency situation, you will lose only those changes you made to a shared file after you issued ?ESFF (provided the ?ESFF completes without error).

Who Can Use It?

There are no special process privileges needed to issue this call, and there are no restrictions concerning file access.

What It Does

?ESFF flushes modified shared pages to disk. Even when an I/O error occurs on one or more of the modified pages, the operating system tries to flush the remaining pages.

When an ?ESFF completes successfully, all pages that were modified at the start of the ?ESFF are on disk. However, if another process modifies a page after the ?ESFF begins, that page may or may not be flushed to disk when the ?ESFF completes.

Note, however, that ?ESFF does not ensure that you'll be able to access the flushed pages; use ?UPDATE to write out updated file descriptor information for the shared file.

Notes

- See the description of ?UPDATE in this chapter.

AOS/VS

?EXEC *[packet address]*
error return
normal return

Input

AC0	Reserved (Set to 0.)
AC1	Host ID for Resource Management Agent (RMA) deflection, otherwise 0.
AC2	Address of the ?EXEC packet, unless you specify the address as an argument to ?EXEC

Output

AC0	Undefined
AC1	Undefined
AC2	Address of the ?EXEC packet

Error Codes in AC0

EREGN	End of get next sequence
ERRBO	(Request) refused by operator
ERVBP	Invalid byte pointer passed as a system call argument
ERVWP	Invalid word pointer passed as a system call argument
ERWMT	Volume not mounted
ERXNA	EXEC not available
ERXSP	Invalid EXEC string parameter
ERXUF	EXEC request function unknown (You passed an unknown value in offset ?XRFNC.)

Why Use It?

You can use ?EXEC to request a service, such as mounting a labeled tape, from the EXEC utility.

Who Can Use It?

One special privilege needed to issue this call is the batch usage privilege (value ?PBCHPRV) for submitting batch jobs (see the section "Queuing a File Entry"). The only other special privileges needed are Write access to a queue to submit it, and Read access to a queue to issue a QDISPLAY command against it. There are no restrictions concerning file access.

What It Does

?EXEC directs the EXEC utility to perform one of the following functions on behalf of the calling process:

- Assign or cancel a logical name to a tape unit or non–LD disk (an uninitialized disk that you want to treat as a whole unit) and issue an operator mount or dismount message.
- Mount an unlabeled tape. (See Figures 2–22 and 2–23, on Pages 2-106 and 2-107respectively.)
- Mount a labeled tape. (See Figures 2–24 and 2–25, on Pages 2-107and 2-108respectively.)
- Dismount a labeled or unlabeled tape. (See Figure 2–26, Page 2-109.)
- Back up your files. (See Figure 2–27, Page 2-110.)
- Place a request into a queue. (See Figures 2–28, 2–29, and 2–30, Pages 2-113, 2-123, and 2-124respectively.)
- Hold, unhold, or cancel a queue request. (See Figure 2–31, Page 2-127.)
- Provide an EXEC status report. (See Figures 2–32 and 2–33, Pages 2-129 and 2-130.)
- Submit a job to a MOUNT queue. (See Figure 2–34, Page 2-131.)
- Dismount a unit (extended request). (See Figure 2–35, Page 2-132.)
- Modify the queuing parameters of a queued job. (See Figure 2–36, Page 2-133.)
- Obtain one or more queue names. (See Figure 2–37, Page 2-137.)
- Obtain QDISPLAY information, given a queue name. (See Figure 2–38, Page 2-139.)

Each of these functions takes a unique packet. However, the first offset, ?XRFNC, which defines the action to be performed, is common to all the packets. The second offset, ?XPRV, is also common; it contains the packet revision number.

Mounting Unlabeled and Labeled Tapes

If you use ?EXEC to mount a tape (labeled or unlabeled), you must specify an operator message string somewhere in your program.

Offsets ?XMUT and ?XMLT are byte pointers to the operator message text string. The operator message cannot exceed 80 characters, including the delimiter. The delimiter can be New Line, carriage return, form feed, or null.

The operator message should contain at least some of the following information:

- The name of the tape reel (if it is not the volume name)
- Where the tape reel is stored
- The type of tape unit to use with the tape reel (for example, high–density, 7–track, or 9–track)
- Any special instructions to the operator

The logical name of the tape, to which offset ?XMUL or ?XMLL points, must be a simple unique name. EXEC takes this name and creates a link in the caller's initial working directory (as specified in the user's profile) with the pathname logon–dir:<LOGICALNAME>. This link resolves to @MTXn for unlabeled tape mounts and to @LMT:VOLID for labeled tape mounts. The X of @MTXN can be A, B, C, or D depending on the tape drive.

To mount an unlabeled tape (called a unit mount), you must set the first word of the ?EXEC packet, ?XRFNC, to ?XFMUN or ?XFXUN. (See Figure 2–22 and Figure 2–23.)

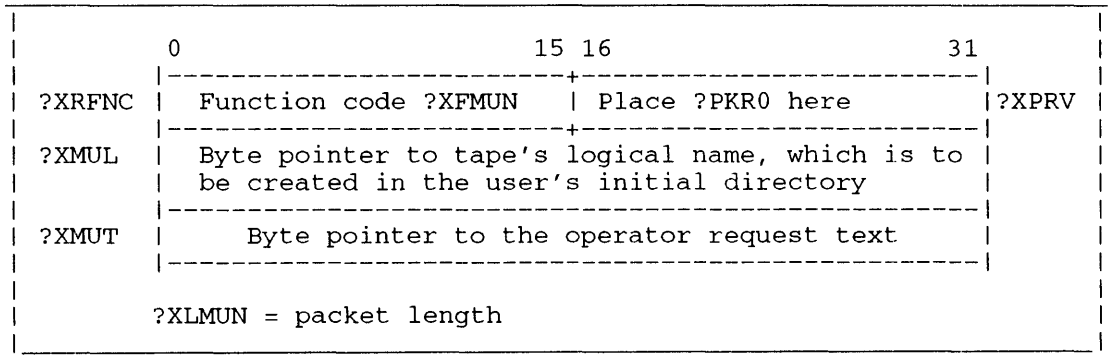


Figure 2–22. Structure of ?EXEC Packet for Unlabeled Mount Function ?XFMUN

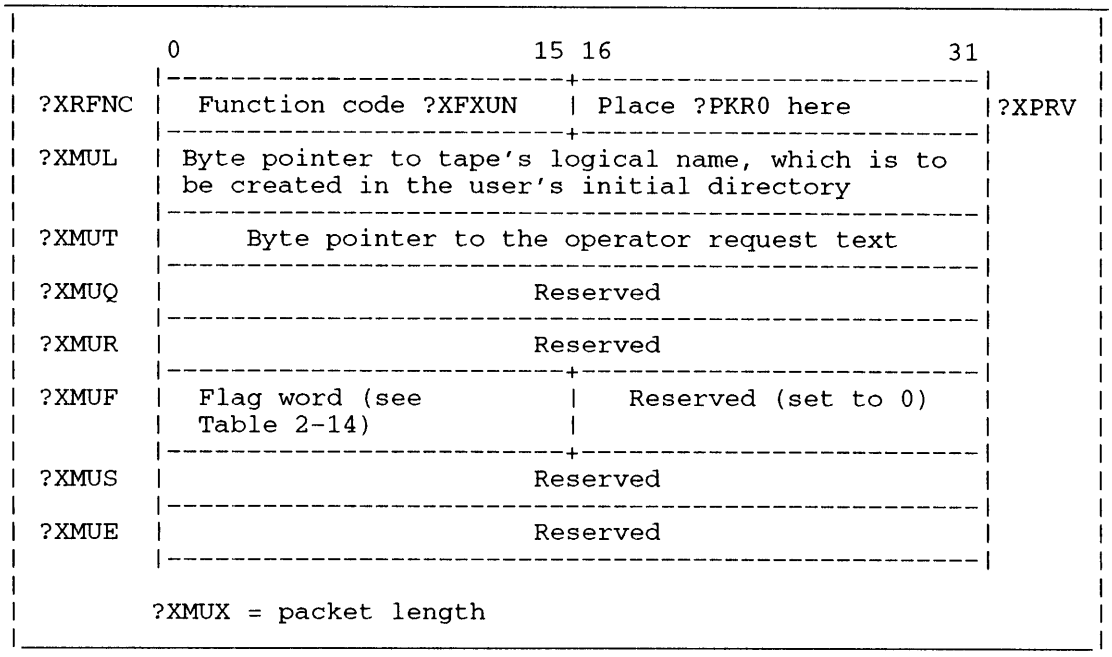


Figure 2–23. Structure of ?EXEC Extended Packet for Unlabeled Mount Function ?XFXUN

To mount a labeled tape, you must set ?XRFNC to ?XFMLT or ?XFXML. (See Figure 2–24 and Figure 2–25.)

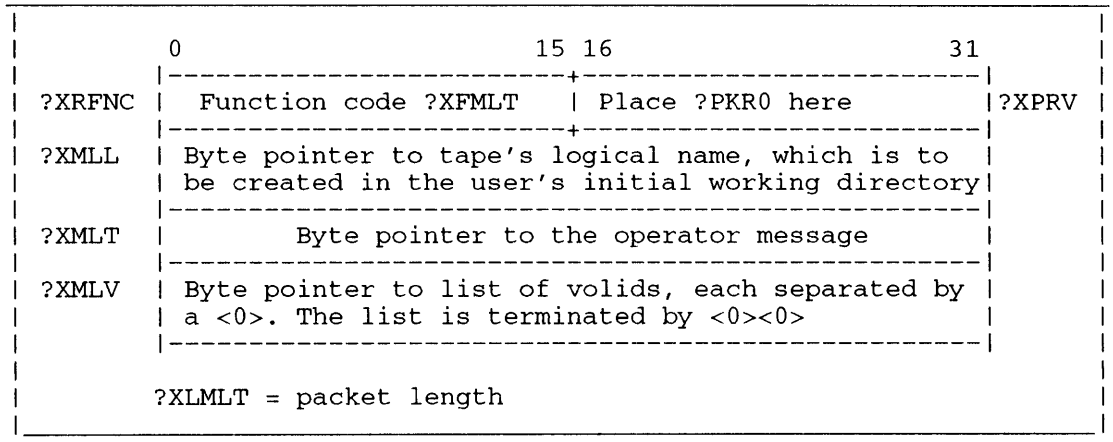


Figure 2–24. Structure of ?EXEC Packet for Labeled Mount Function ?XFMLT

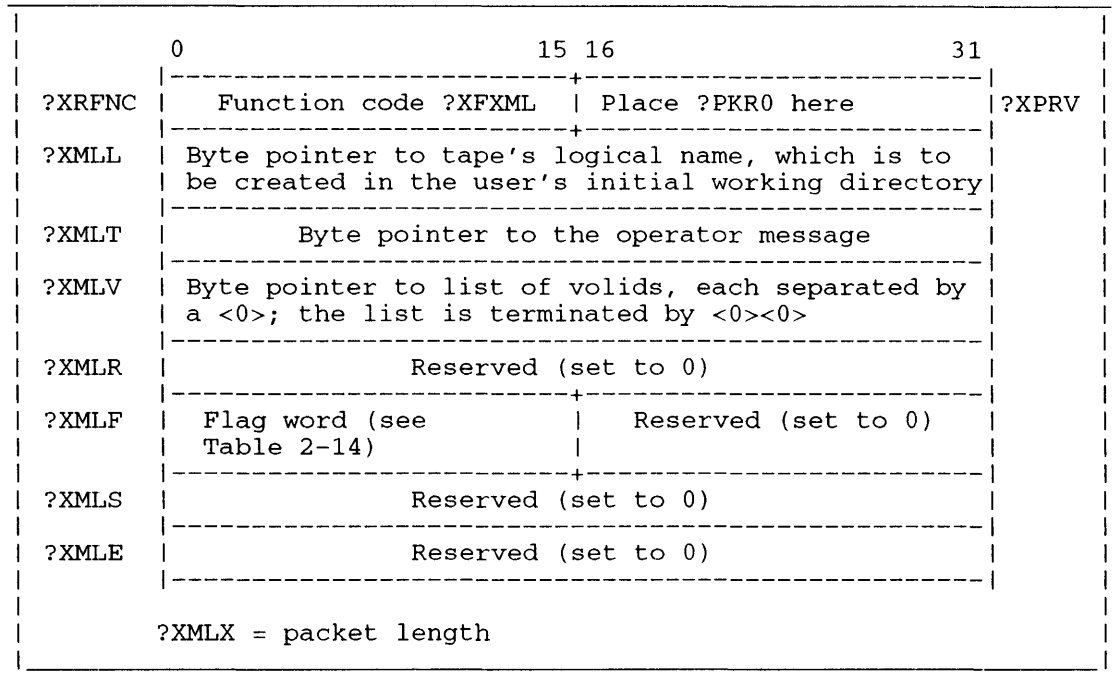


Figure 2–25. Structure of ?EXEC Extended Packet for Labeled Mount Function ?XFXML

Table 2–14. Flags for EXEC Functions ?XFXUN and ?XFXML

Flag	Description
?XMFC	If set, checks volid that operator specified against tape.
?XMFI	If set, the tape is in IBM format.
?OPD0 ?OPD1 ?OPD2	Density to use. This applies to tape drives capable of software selection of the density only. (See the description of the ?OPEN system call for information on setting the bits.)
?XMFR	If set, EXEC assumes that the tape to be mounted is to be read only and sets the ACL to USER,RE.
?XMEL	If set, allows you to extend the volid list. This applies to labeled tape mount requests only.

Dismounting Unlabeled and Labeled Tapes

To dismount either type of tape, you must set ?XRFNC to ?XFDUN. (See Figure 2–26.)

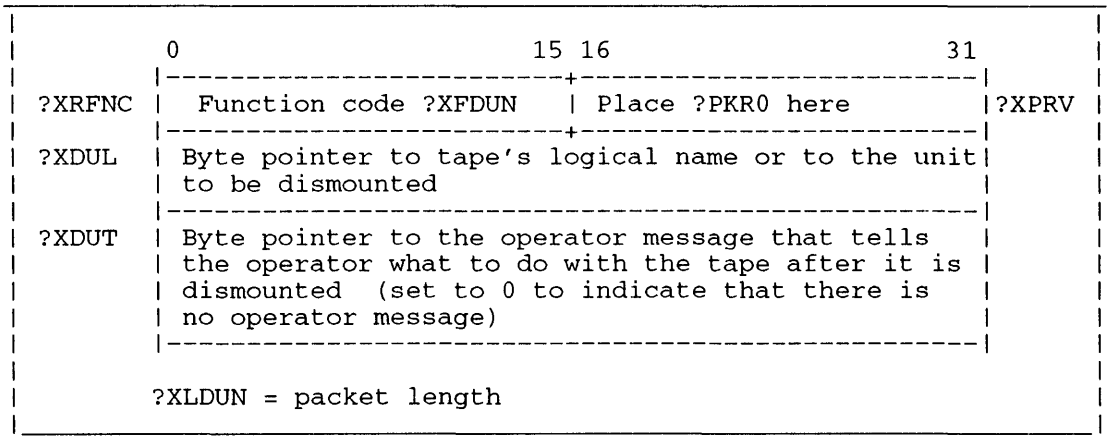


Figure 2–26. Structure of ?EXEC Packet for Dismounting a Tape, ?XFDUN

If you use ?EXEC to dismount a tape, an operator message is optional. (See "Mounting Unlabeled and Labeled Tapes" in this description for more information on operator messages.)

The number of characters in the operator message pointed to by offset ?XDUT cannot exceed 80 characters, including the terminator. (This is also true of the message pointed to by offset ?XMUT for unlabeled tapes. See Figure 2–22 and Figure 2–23.)

Backing Up Your Files

If you decide to back up your files directly instead of using the DUMP_II/LOAD_II and DUMP/LOAD CLI commands, then read this section. This section applies only to 32–bit programs.

Figure 2–27 shows the structure of the ?EXEC packet for backing up files, and Table 2–15 describes the contents of each offset.

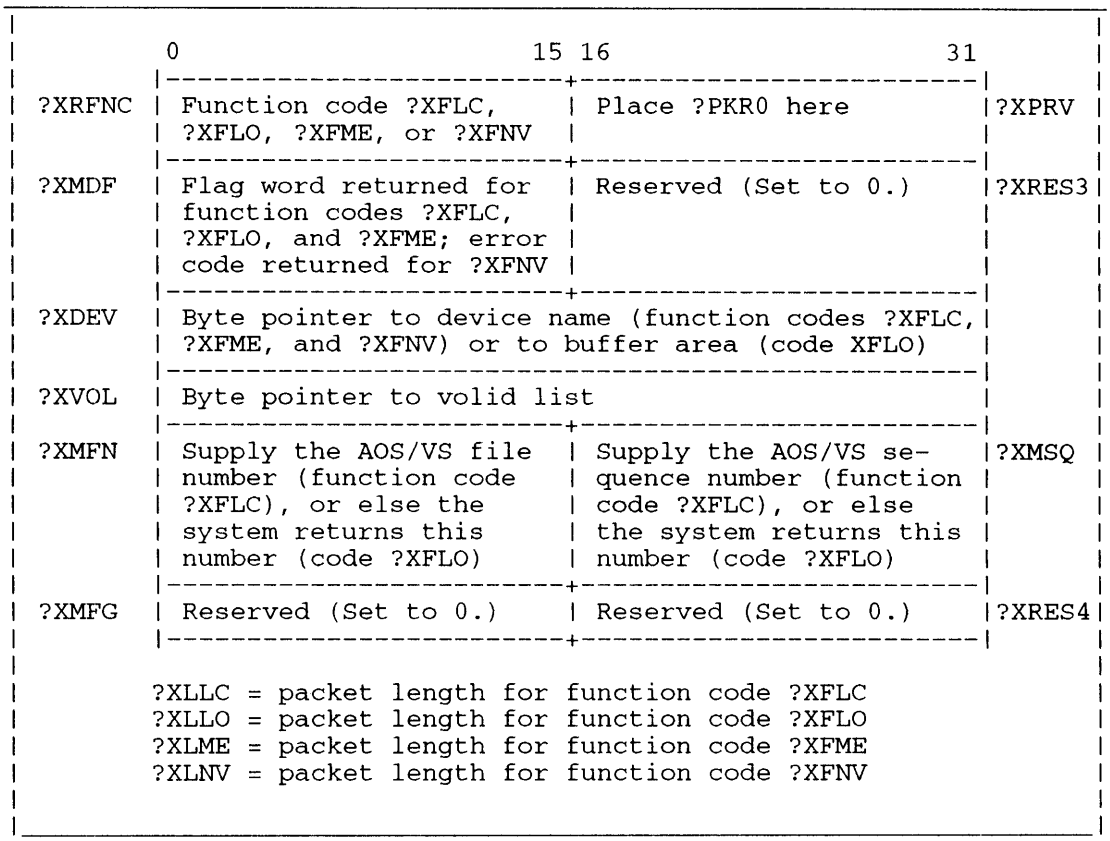


Figure 2–27. Structure of ?EXEC Packet for Tape Backup

Table 2–15. Contents of ?EXEC Packet for Tape Backup*

Offset	Contents
?XRFNC	Function code. Supply one of the following: ?XFLC -- Labeled tape close. ?XFLO -- Labeled tape open. ?XFME -- Mount error. ?XFNV -- Labeled tape next volume.
?XPRV	Packet revision number. Supply ?PKR0.
?XMDF	If you supply ?XFLC, ?XFLO, or ?XFNV to offset ?XRFNC, then AOS/VS returns the following status bit definitions in this flag word: ?XMFS -- First/specific volume. ?XMSQB -- This is a sequential operation. ?XSCV -- Volid not verified yet. ?XSD1 -- Density bits. ?XSD2 -- Density bits. ?XSDEN -- Density bits. ?XSEL -- Extended volid list. ?XSIBM -- IBM tape. ?XSMT -- Explicitly mounted labeled tape. ?XSRO -- Read only. ?XSVU -- Labeled tape volid is unknown. If you supply ?XFME to offset ?XFRNC, then AOS/VS returns an error code.
?XRES3	Reserved for EXEC. (Set to 0.)
?XDEV (doubleword)	If you supply ?XFLC, ?XFME, or ?XFNV to offset ?XRFNC, then supply a byte pointer to the actual device name. If you supply ?XFLO to offset ?XRFNC, then supply a byte pointer to a buffer area; AOS/VS returns the device name in the buffer.
?XVOL (doubleword)	Supply a byte pointer to the volid list.
?XMFN	This offset is the AOS/VS file number. It also supports the sequential functionality to bypass the tape rewind operations between sequential file operations. Supply the file number when offset ?XRFNC contains ?XFLC, wait for AOS/VS to return the file number when offset ?XRFNC contains ?XFLO, or supply 0 when offset ?XRFNC contains ?XFME or ?XFNV.
?XMSQ	This offset is the AOS/VS sequence number. It also supports the sequential functionality to bypass the tape rewind operations between sequential file operations. Supply the file number when offset ?XRFNC contains ?XFLC, wait for AOS/VS to return the file number when offset ?XRFNC contains ?XFLO, or supply 0 when offset ?XRFNC contains ?XFME or ?XFNV.
?XMFG	Reserved for EXEC. (Set to 0.)
?XRES4	Reserved for EXEC. (Set to 0.)

* There is no default unless otherwise specified. (continued)

Queuing a File Entry

The ?EXEC packet for placing a queue request allows you to queue a file for batch processing or for spooled output to one of the following devices:

- Line printer (If you want to print a file in true binary mode or to specify 8–bit characters in the input file or to have no header line, then pay special attention to offsets ?XPRV and ?XFG2 in Figure 2–28 and Table 2–16.)
- Terminal and other communications devices
- Digital plotter
- File transfer agent (FTA)
- SNA

Figure 2–28 shows the structure of the ?EXEC packet for queue requests, and Table 2–16 describes the contents of each offset. Note that some of the packet options correspond to CLI commands for the EXEC utility.

The first offset in the packet, ?XRFNC, defines the kind of queue. Other offsets allow you to postpone processing the queue request, assign the job an EXEC priority, delete the file after processing, and select several print options (for example, print the file's pathname, print page numbers, fold long lines, and so forth).

Offset ?XTYP points to the queue name of the desired queue. This name must correspond to the queue type chosen in offset ?XRFNC, unless you selected ?XFSUB (for any type of queue except ?XFUSR).

Queue names resemble device names, but they do not begin with the @ prefix. For example, if you have one or more spooled line printers, one of the line printer queue names should be LPT. The system manager usually sets the queue names for an installation.

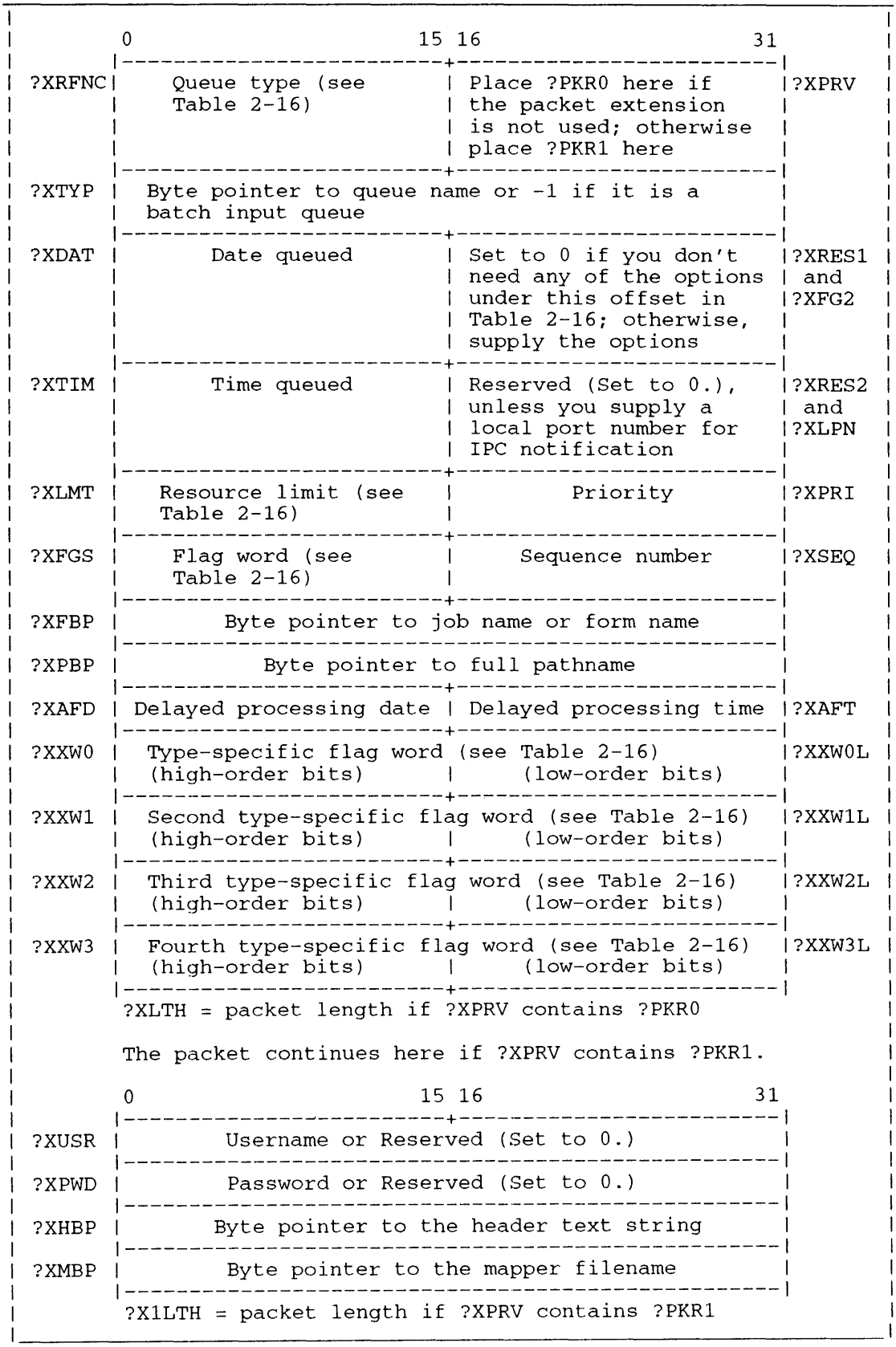


Figure 2–28. Structure of ?EXEC Packet for Queue Requests

Table 2–16. Contents of ?EXEC Packet for Queue Requests*

Offset	Contents
?XRFNC	Queue type: Place entry into one of the following: ?XFBAT--Batch queue. ?XFFTA--FTA queue. ?XFHAM--HAMLET queue. ?XFLPT--Print queue. ?XFPLT--Plot queue. ?XFSNA--SNA/RJE queue. ?XFSUB--Any type of queue (except ?XFUSR). ?XFOTH--Submit job for another user. Setting Bit 0 of ?XRFNC indicates that RMA should deflect this request to the remote host specified in AC1. A ?XFBAT request is, other than the value in offset ?XRFNC, identical to a generic ?XFSUB queue submission call. Function code ?XFBAT is provided to allow for further specialization and expansion of batch queue submission.
?XPRV	Packet revision number. (Set to ?PKR1 if offset ?XRES1/?XFG2 does not contain 0 or if ?XRFNC contains ?XFOTH; otherwise, set to ?PKR0.)
?XTYP (doubleword)	Byte pointer to queue name (must correspond to type in ?XRFNC).
?XDAT	Date queued (returned by the OS in standard date notation).

* There is no default unless otherwise specified. (continued)

Table 2–16. Contents of ?EXEC Packet for Queue Requests*

Offset	Contents
?XRES1 and ?XFG2	If no options are needed, set to 0. If ?XRFNC is equal to ?XFLPT or ?XFOTH, the options are: ?BMCO = 1B(?XFCO) -- Print output in 2 columns. ?BMNH = 1B(?XFNH) -- Print without a header page. ?BM8B = 1B(?XF8B) -- The file might contain 8-bit characters, and the high order bit is significant. If ?BM8B is not set then EXEC prints the file according to the operator's CONTROL @EXEC START command. ?BMTB = 1B(?XFTB) -- Print the file in true binary mode. This means that all <377> characters are passed to the printer. This mode is the same as PASSTHRU mode. In contrast, binary mode (?BMBI value) will pass on only the second of the two-character sequence <377><xxx> where <xxx> is any value between <000> and <377>, inclusive. In general, binary mode (?BMBI value) ignores single <377> characters. ?BMEP = 1B(?XFEP) --If the offset ?XRFNC contains ?XFOTH, this bit is set to indicate that the password is encrypted. If queue type is submitted for ?XFOTH, then ?XUSR is a byte pointer to username. If Superuser is also on, ?XPWD can be 0 or a byte pointer to password.
?XTIM	Time queued (returned by the OS in standard time notation).
?XRES2 and ?XLPN	Reserved for EXEC. (Set to 0.) Exception: If you want EXEC to send (via ?ISEND) an IPC message for notification, then supply the local port number and specify ?BMCN in offset ?XFGS. Figure 2-29 contains the format of the IPC message.
?XLMT	Resource limit (enforced only if operator enabled the EXEC limit feature): For print queue: ?XLMT = maximum number of pages to be printed. If you default, EXEC estimates the number of pages as follows: pages = (bytes/1000) + 4 where bytes = total number of bytes in file.

* There is no default unless otherwise specified. (continued)

Table 2–16 .Contents of ?EXEC Packet for Queue Requests*

Offset	Contents
?XLMT (continued)	For batch queue: ?XLMT = maximum number of CPU (including the CLI and all other processes created by this batch job). If you default, EXEC estimates a value of 30 biseconds; that is, 1 minute. DEFAULT = 0 (see above)
?XPRI	Entry's priority, within the range: n (highest) through 255 (lowest). where n = maximum priority specified in your user profile. DEFAULT = -1 (EXEC assigns a priority in the middle of the range and returns the value here).
?XFGS	Flag word ?BMSH = 1B(?XFSH)--Hold sequence number. ?BMDA = 1B(?XFDA)--Delete file specified in offset ?XPBP after processing. (Caller must have Owner or Write access to the file.) ?BMNR = 1B(?XFNR)--Do not restart entry if EXEC or the system crashes during processing. ?BMPE = 1B(?XFPE)--Pend the calling process until the operator acts on the request. 0B(?XFPE)--Don't pend the calling process, but still receive confirmation when the operator acts on the request.

* There is no default unless otherwise specified. (continued)

Table 2–16 . Contents of ?EXEC Packet for Queue Requests*

Offset	Contents
?XFGS (continued)	?BMBI = 1B(?XFBI)--Output is binary. ?BMOP = 1B(?XFOP)--Start processing only if operator is on duty. (We recommend setting this flag for batch jobs that require tape mounting or dismounting.) ?BMNO = 1B(?XFNO)--Notify the user's terminal when this entry is completed. (The message is lost if you log off, and then log on again; there is no message if you are not a descendant of EXEC.) ?BMRA = 1B(?XFRA)--Determines the value of offsets ?XAFD and ?XAFT in this packet. Offsets ?XAFD and ?XAFT are relative. ?BMTI = 1B(?XFTI)--Print titles on output listing. ?BMFO = 1B(?XFFO)--Fold long lines. DEFAULT = 0 (no flags).

* There is no default unless otherwise specified. (continued)

Table2–16. Contents of ?EXEC Packet for Queue Requests*

Offset	Contents
?XSEQ	Sequence number EXEC assigned to this entry (returned by EXEC).
?XFBP (doubleword)	For batch queue: byte pointer to job name. For other queue type: byte pointer to form name. If you set this to 0, EXEC will use the device's default forms. (You must have previously defined the form filename and must have access to it.)
?XPBP (doubleword)	Byte pointer to full pathname (starting with the root) of the file to be processed. For FTA queues: byte pointer to full pathname of the source file.
?XAFD and ?XAFT	Interpretation depends on flag bit ?BMRA in offset ?XFGS. If ?BMRA was not set: ?XAFD - read as the date (in file system format) after which the entry can be processed. ?XAFT - read as the time (in file system format) after which the entry can be processed. If ?BMRA was set ?XAFD - read as a number of whole days after which the entry can be processed. ?XAFT - read as a number of bi-seconds (2-second intervals) after which the entry can be processed. If ?BMRA was set, EXEC adds the time periods specified in ?XAFD and ?XAFT, starting from the time the queue request was made, and begins processing the entry after the designated time period has elapsed. The default value for ?XAFD and ?XAFT is 0 (no delayed processing).

* There is no default unless otherwise specified. (continued)

Table 2–16. Contents of ?EXEC Packet for Queue Requests*

Offset	Contents
?XXW0 (sometimes doubleword)	Type-specific flag word For print queue (single word): - If ?XXW0 is nonzero, start printing on the page number given here. - If ?XXW0 is 0 (the default), start printing on the first page. For batch queue (doubleword): - ?XXW0 is the high-order portion of a byte pointer to the pathname of the file you want to use as the @OUTPUT file. This file will be created if it does not already exist (EXEC does not print or delete this file). - If XXW0 is 0, EXEC creates, prints, and then deletes the Default Generic Output File, :QUEUE:USER.OUTPUT.SEQ. (If this file is empty, printing is omitted.) For FTA queue (single word): stream number if desired, otherwise 0. For all other queue types (single word): ?XXW0 must be 0. DEFAULT = 0 (no flags).
?XXW0L	For batch queue: ?XXW0L contains the low-order bits of ?XXW0. For all other queue types (single word): ?XXW0L must be 0. DEFAULT = 0.

* There is no default unless otherwise specified. (continued)

Table 2–16. Contents of ?EXEC Packet for Queue Requests*

Offset	Contents
?XXW1 (sometimes doubleword)	Second type-specific flag word. For print queue (single word): If ?XXW1 is nonzero, stop printing at the page specified here. For batch queue (doubleword): ?XXW1 is a byte pointer to generic @LIST file; otherwise, contains 0 (for default @LIST file). For FTA queue (doubleword): byte pointer to full pathname of the destination file. For all other queue types (single word): ?XXW1 must be 0.
?XXW1L	For batch and FTA queues: ?XXW1L contains the low-order bits of ?XXW1. For all other queue types: ?XXW1L must be 0.
?XXW2	Third type-specific flag word. For batch queue: ?XXW2 must be 0. For SNA queue: Indicate the type of read by choosing one of the following mutually exclusive flags: ?X2LN - data-sensitive line read. ?X2SE - dynamic (sequential) read. ?X2EB - dynamic read of 80 characters. Optionally, indicate data translation: ?X2HO - Hollerith to EBCDIC translation. Specify the class of device with one of the following (these are mutually exclusive): ?X2CD - card. ?X2CN - terminal. ?X2EX - exchange.

* There is no default unless otherwise specified. (continued)

Table 2–16. Contents of ?EXEC Packet for Queue Requests*

Offset	Contents
?XXW2 (continued)	For FTA queue: Set one or more of the following transfer option flags: ?X2CM - compression requested. ?X2RC - process only if source more recent. ?X2AP - append source to destination if it exists. ?X2SD - delete source after transfer. ?X2DD - delete destination before transfer. ?X2RM - use record mode transfer. ?X2TO - let connection time out after failure. ?X2CP - restart at last checkpoint after failure. For all other queue types: ?XXW2 equals number of copies the EXEC spooler creates. (If ?XXW2 equals 0, one copy is assumed.)
?XXW2L	Reserved (Set to 0.)
?XXW3 (sometimes doubleword)	For print and batch queues (doubleword): ?XXW3 contains a byte pointer to a text string (destination). This string appears in block letters at the top of any header or trailer pages. If ?XXW3 contains 0, the text string is the default username. However, if you supply ?PKR1 in offset ?XPRV you must supply 0 in this doubleword. The byte pointer to the text string is in offset ?XHBP of the packet extension. For an SNA queue, set one of these mutually exclusive flags: ?X3TR - transparent transmission. ?X3CO - concatenate. For all other queue types (single word): ?XXW3 must be 0.
?XXW3L	For print and batch queues: ?XXW3L contains the low-order bits of ?XXW3. For all other queue types: ?XXW3L must be 0.

* There is no default unless otherwise specified. (continued)

Table 2–16. Contents of ?EXEC Packet for Queue Requests*

Offset	Contents
?XUSR (doubleword)	For ?XRFNC equal to ?XFOTH, this field is username, otherwise it is reserved. (Set to 0.)
?XPWD (doubleword)	For ?XRFNC equal to ?FOTH, this field is password, otherwise it is reserved. (Set to 0.)
?XHBP (doubleword)	For print and batch queues ?XHBP contains a byte pointer to a text string (document name). This string appears in block letters in the middle of any header or trailer pages. If ?XHBP contains 0, the text string is the default document name.
?XMBP (doubleword)	For print and batch queues ?XMBP contains a byte pointer to a mapper filename. A mapper file alters characters, such as changing lowercase characters to uppercase characters for printing. Supply 0 if there is no mapper file.

* There is no default unless otherwise specified. (concluded)

NOTE: Supply values for the following four offsets only if offset ?XPRV contains ?PKR1.

For more information on the format of the ?XXW2 and ?XXW3 offsets, refer to the parameter files PARU.32.SR and PARU.16.SR. The sections describing the ?EXEC parameters are particularly useful when issuing queue submissions for FTA and SNA queues.

If you place ?PKR1 in offset ?XPRV and supply a local port number in offset ?XRES2/?XLPN and set Bit ?XFCN in offset ?XFGS, then ?EXEC sends an IPC message to the local port. The recipient can then notify a user that a print request is being processed.

The IPC comes from the same global port as that of an ?ILKUP of the specified queue in :PER.

All queue files in :PER have the same global port number. To identify the queue associated with the notification ?EXEC will return the queue name in the IPC message. The format of the IPC message that ?EXEC returns is in Figure 2–29.

If the operator has set the EXEC limit feature, which limits the number of pages or the CPU time allowed for your job, you must set offset ?XLMT accordingly, or default it to 0. When you default ?XLMT, EXEC sets the appropriate limit.

Offset ?XFBP is a byte pointer to the job name for batch queue requests and to the form name for other queue requests. (If you set this to 0, EXEC uses the device's previously defined default forms.) The job name, which must contain from 1 through 31 legal filename characters, identifies a batch job (one or more programs submitted as a unit to batch). The job name you select need not be unique.

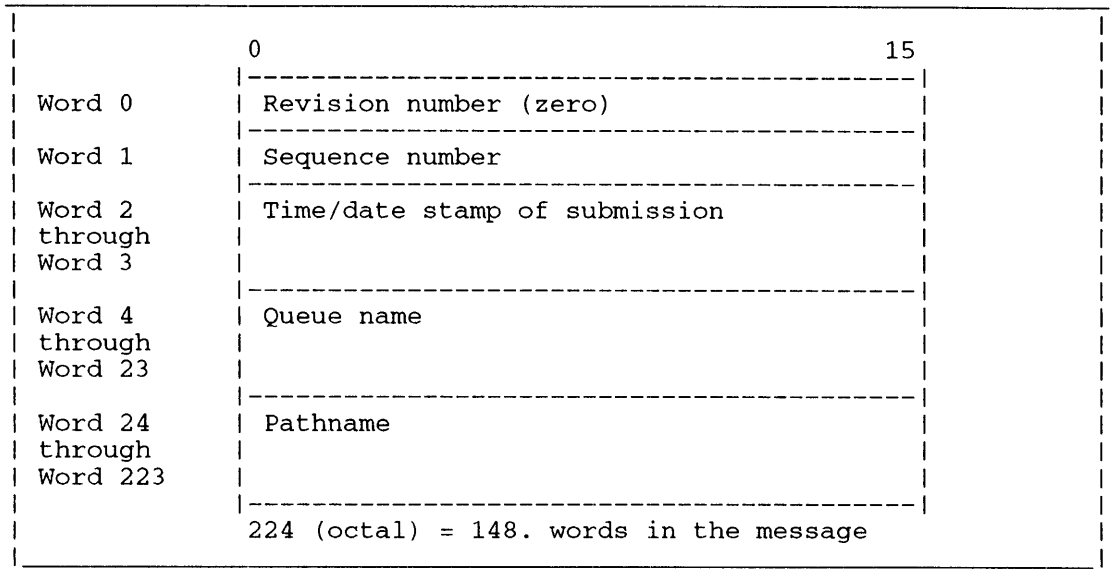


Figure 2–29. Structure of the IPC Print Notification Message from ?EXEC

Form name is the name of a form in the :UTIL:FORMS directory that was created with the CLI forms control utility (FCU). Typically, the system manager creates files in :UTIL:FORMS. Like the job name, the form name must contain from 1 through 31 legal filename characters. To determine the valid form names for your installation, type

ACL/V :UTIL:FORMS:+

This returns the names of the form files in :UTIL:FORMS and their corresponding ACLs. You can use any form file for which you have Read access.

Offsets ?XAFD and ?XAFT allow you to delay processing the queue request until a specific time has elapsed. The interpretation of these offsets varies, depending on whether you set flag ?XFRA in offset ?XFGS.

If you set the ?XFRA flag, you must set offsets ?XAFD and ?XAFT to the desired date and time offsets from the current time. If you do not set the ?XFRA flag, set offsets ?XAFD and ?XAFT to the desired processing date and time in the standard notations.

Offset ?XXW2 is a print flag with different meanings for different queue types. For a batch queue, this flag must be 0. For an FTA queue you can select any combination of transfer options flags. An SNA queue has a number of options associated with it, several of which are mutually exclusive. You can choose ONE read type (data–sensitive, dynamic, or dynamic 80–character) and ONE class of device (card, terminal, or exchange). You also have the option of indicating Hollerith to EBCDIC translation with the ?X2HO flag.

For any other queue, ?XXW2 is the number of copies the EXEC spooler creates (0 is interpreted as one copy.)

Queuing a User–Cooperative Job

Use the ?XFUSR subpacket of the ?EXEC system call to place a job request in a user–cooperative queue. Figure 2–30 shows the format for the ?XFUSR subpacket. Table 2–17 describes the contents of the ?XFUSR subpacket.

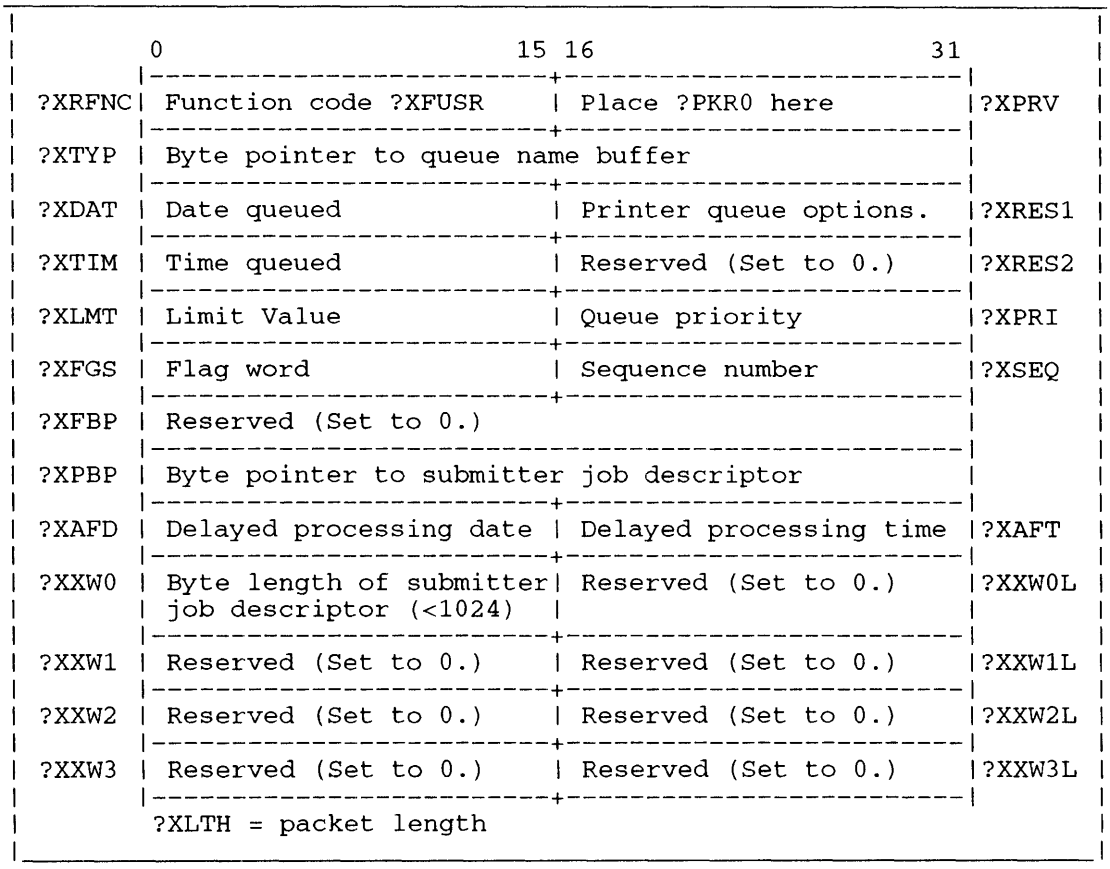


Figure 2–30. Structure of ?XFUSR Subpacket of ?EXEC System Call

Table 2–17. Contents of ?XFUSR Subpacket of ?EXEC System Call

Offset	Contents
?XRFNC	?XFUSR Setting Bit 0 of ?XRFNC indicates that RMA should deflect this request to the remote host specified in AC1.
?XPRV	Packet revision number; set to ?PKR0.
?XTYP	Byte pointer to the queue name of the user queue.
?XDAT	Date queued (returned by the operating system in standard date notation).
?XTIM	Time queued (returned by the operating system in standard date notation).
?XLMT	Resource limit; see Table 2-16.
?XPRI	Priority of entry within the range of n through 255. The value of n equals the maximum priority specified in your user profile. DEFAULT = -1 (EXEC assigns a priority in the middle of the range and returns the value here).
?XFGS	Flag word ?XFSH = Set sequence number held (/HOLD) by user flag for this entry. ?XFDA = Ignored. ?XFNR = Do not restart entry (/NORESTART) if EXEC or the system crashes during processing. ?XFBI = Ignored. ?XFOP = Start processing only if operator is on (/OPERATOR) duty. ?XFNO = Notify (/NOTIFY) the user's terminal when this entry is completed. The message is lost if /NRM terminal characteristic is on or if you log off, and then log on again. There is no message if you are not a descendant of EXEC.

(continued)

Table 2–17. Contents of ?XFUSR Subpacket of ?EXEC System Call

Offset	Contents
?XFGS (continued)	?XFPE = Pend (/PEND) the calling process until the operator acts on the request. ?XFRA = Begin the job after (/AFTER=+) the specified date and time. ?XFRA determines how to interpret the values in offsets ?XAFD and ?XAFT. ?XFTI = Ignored. ?XFFO = Ignored. DEFAULT = 0 (no flags).
?XSEQ	The sequence number that EXEC assigns to this entry and returns here.
?XPBP	Byte pointer to the submitter part of the job descriptor. A job descriptor completely identifies a job. The cooperative defines the form and content of the submitter part. Refer to the "Programming EXEC Cooperative Processes" manual for more information on the job descriptor.
?XAFD and ?XAFT	Interpretation depends on flag bit ?XFRA in offset ?XFGS. If ?XFRA is not set: ?XAFD - read as the date (in file system format) after which the entry can be processed. ?XAFT - read as the time (in file system format) after which the entry can be processed. If ?XFRA is set: ?XAFD - read as a number of whole days after which the entry can be processed. ?XAFT - read as a number of bi-seconds (2-second intervals) after which the entry can be processed.
?XAFD and ?XAFT (continued)	If ?XFRA was set, EXEC adds the time periods specified in ?XAFD and ?XAFT, starting from the time the queue request was made, and begins processing the entry after the designated time period has elapsed. The default value for ?XAFD and ?XAFT is 0 (no delayed processing).
?XXW0	Byte length of the submitter part of the job descriptor. It must be less than 1024 bytes.

(concluded)

Using Hold, Unhold, Cancel Queue Requests (AOS/VS)

You can also use ?EXEC to suspend (hold), release (unhold), or clear (cancel) a queue entry you have created. An unhold directive negates a previously issued hold directive. Figure 2–31 shows the structure of the AOS/VS packet for the hold, unhold, and cancel directives. Table 2–18 lists the AOS/VS ?EXEC packet's contents.

As in the other ?EXEC packets, the value of offset ?XRFNC determines the action you want to perform: ?XFHOL holds the specified entry, ?XFUNH releases the entry, and ?XFCAN cancels the entry. In addition to the ?XFHOL specification, each queue request has one hold bit that only the operator process can access. If any hold bit is set, EXEC does not select the entry for processing.

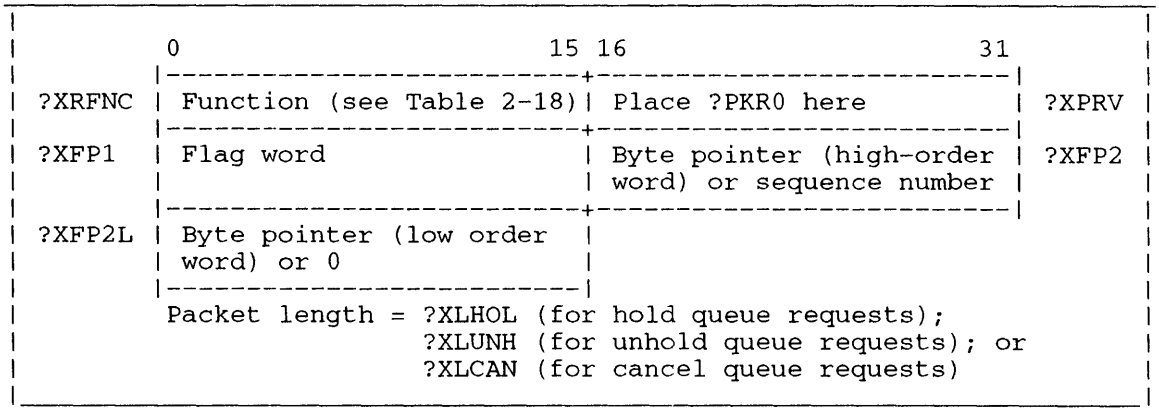


Figure 2–31. Structure of AOS/VS ?EXEC Packet for Hold, Unhold, or Cancel Queue Requests

Offset ?XFP2 contains either the high–order word of a byte pointer to the entry's job name or else the entry's sequence number.

If you use offsets ?XFP2/?XFP2L as a byte pointer, EXEC searches only the batch queue, and changes only the status of the jobs which have this job name. If these offsets contain a sequence number (?XFP2L contains zero), EXEC searches all queue entries and changes the hold bit of the entry designated by the specified sequence number.

When you select the Cancel option (?XFCAN), EXEC clears all hold bits, including the operator bits, and sets the entry's Canceled by User bit.

Table 2–18. Contents of AOS/VS ?EXEC Packet for Hold, Unhold, or Cancel Queue Requests*

Offset	Contents
?XRFNC	Function: select one of the following values: ?XFHOL--Suspend (hold) the specified entry. ?XFUNH--Release (unhold) the specified entry. ?XFCAN--Clear (cancel) the specified entry.
?XPRV	Packet revision number. Place ?PKR0 here.
?XFP1	Flag word - If ?XFP2 and ?XFP2L contain a byte pointer to job name, set to -1. - If ?XFP2 and ?XFP2L contain a sequence number, set to any value.
?XFP2 (doubleword)	Contains one of the following: - Byte pointer to job name. - Sequence number. (Set ?XFP2L to 0.)

* There is no default unless otherwise specified.

Job names and sequence numbers need not be unique; that is, it is possible to have several requests with the same job name or sequence number. In this case, EXEC performs the hold, unhold, or cancel action on all entries that bear the specified sequence number or job name.

Note that the hold and unhold directives do not affect an entry that is currently being processed. The cancel directive will stop the processing of a currently active request.

Obtaining EXEC Status Information

To get selected information about the EXEC process that is currently running, issue ?EXEC with the packet shown in Figure 2–32. The operating system returns the status information to offset ?XFP1 of the packet and, optionally, to the buffer you specify in offset ?XFP2. Table 2–19 describes each offset in the packet.

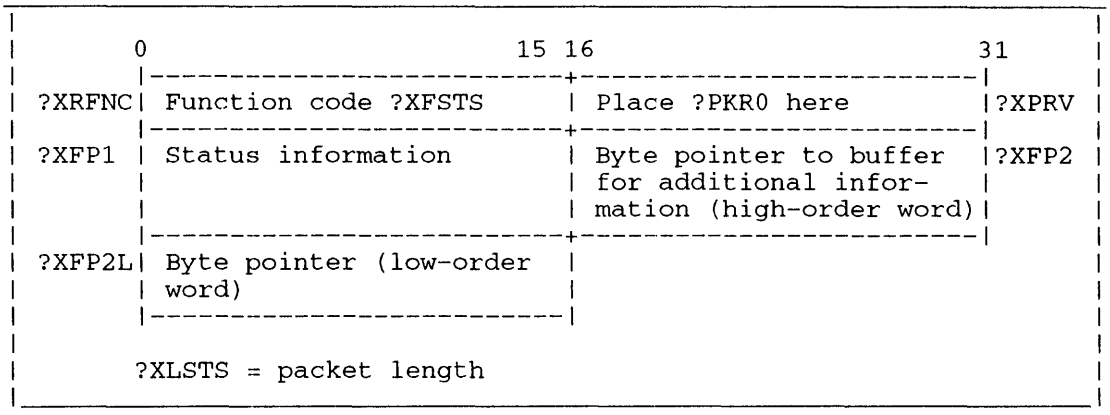


Figure 2–32. Structure of ?EXEC Packet for Status Information

Table 2–19. Contents of ?EXEC Packet for Status Information*

Offset	Contents
?XRFNC	?XFSTS (specifies function: obtain EXEC status information).
?XPRV	Packet revision number. (Set to ?PKR0.)
?XFP1	Status information (returned by EXEC): - If Bit 0 = 1, operator is on duty. - If Bit 0 = 0, system is unattended. - If Bit 1 = 1, your username is logged on in batch mode. - If Bit 1 = 0, your username is logged on at a terminal or is not logged on. - If Bits 8-15 = 0, you are not logged on. - If Bits 8-15 = PID of the process immediately subordinate to EXEC, you are logged on.
?XFP2 (doubleword)	Byte pointer to buffer area you set aside to receive the terminal name or the batch stream name (if you were logged on under EXEC). (The OS returns the terminal name without the @ prefix.) DEFAULT = 0 (no buffer).

* There is no default unless otherwise specified.

Obtaining Extended EXEC Status Information

In addition to the information that function code ?XFSTS returns, you can also obtain the PID of the EXEC process. You do this by specifying function code ?XFXTS in offset ?XRFNC. Issue ?EXEC with the packet shown in Figure 2–33. Table 2–20 describes each offset in the packet.

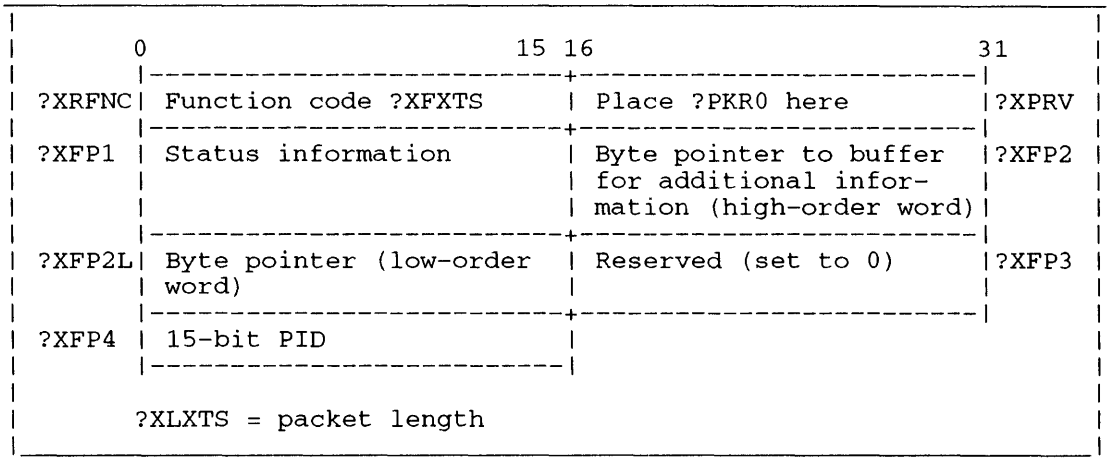


Figure 2–33. Structure of ?EXEC Packet for Extended Status Information

Table 2–20. Contents of ?EXEC Packet for Extended Status Information*

Offset	Contents
?XRFNC	?XFXTS (specifies function: obtain EXEC extended status information).
?XPRV	Packet revision number. (Set to ?PKR0.)
?XFP1	Status information (returned by EXEC): - If Bit 0 = 1, operator is on duty. - If Bit 0 = 0, system is unattended. - If Bit 1 = 1, your username is logged on in batch mode. - If Bit 1 = 0, your username is logged on at a terminal or is not logged on. Bits 2-15 are reserved.
?XFP2 (doubleword)	Byte pointer to buffer area you set aside to receive the terminal name or the batch stream name (if you were logged on under EXEC). (The OS returns the terminal name without the @ prefix.) DEFAULT = 0 (no buffer).
?XFP3	Reserved. (Set to 0.)
?XFP4	If you are logged on, the OS returns the 15-bit PID of the process immediately subordinate to EXEC. If you are not logged on, the OS returns 0.

* There is no default unless otherwise specified.

Submitting a Job to a MOUNT Queue

You use the ?XFMNT function of ?EXEC to submit a job to a MOUNT queue. The CLI MOUNT command uses the function, and it allows specification of queueing parameters such as /HOLD and /AFTER. If you set the ?XFPE flag bit, it indicates that the process should be pended until the operator responds to the queue request (e.g., mounts the tape or refuses the request). If you don't set the ?XFPE flag bit, the process takes the normal ?EXEC call return as soon as the request becomes part of the queue. An operator's response to an unpended mount request (i.e., the ?XFPE flag bit is off) is relayed to the caller so that the caller is aware of the operator's actions.

The ?XFMNT packet in Figure 2–34 allows specification of all the ?XFMUN and ?XFMLT options plus some of the ?XFSUB options. The details of these three options appear respectively in Figure 2–22, Figure 2–24, and Table 2–16/Figure 2–28. You can use the ?XWW1 offset in Figure 2–34 to specify a tape or diskette mount. If you specify neither type of mount, ?EXEC assumes you are issuing a tape mount request.

You can specify a directory location for the link name. In this case, the operating system places in offset ?XWW3 a byte pointer. This byte pointer points to the directory name where the logical name will be created.

Issue ?EXEC with the packet shown in Figure 2–34.

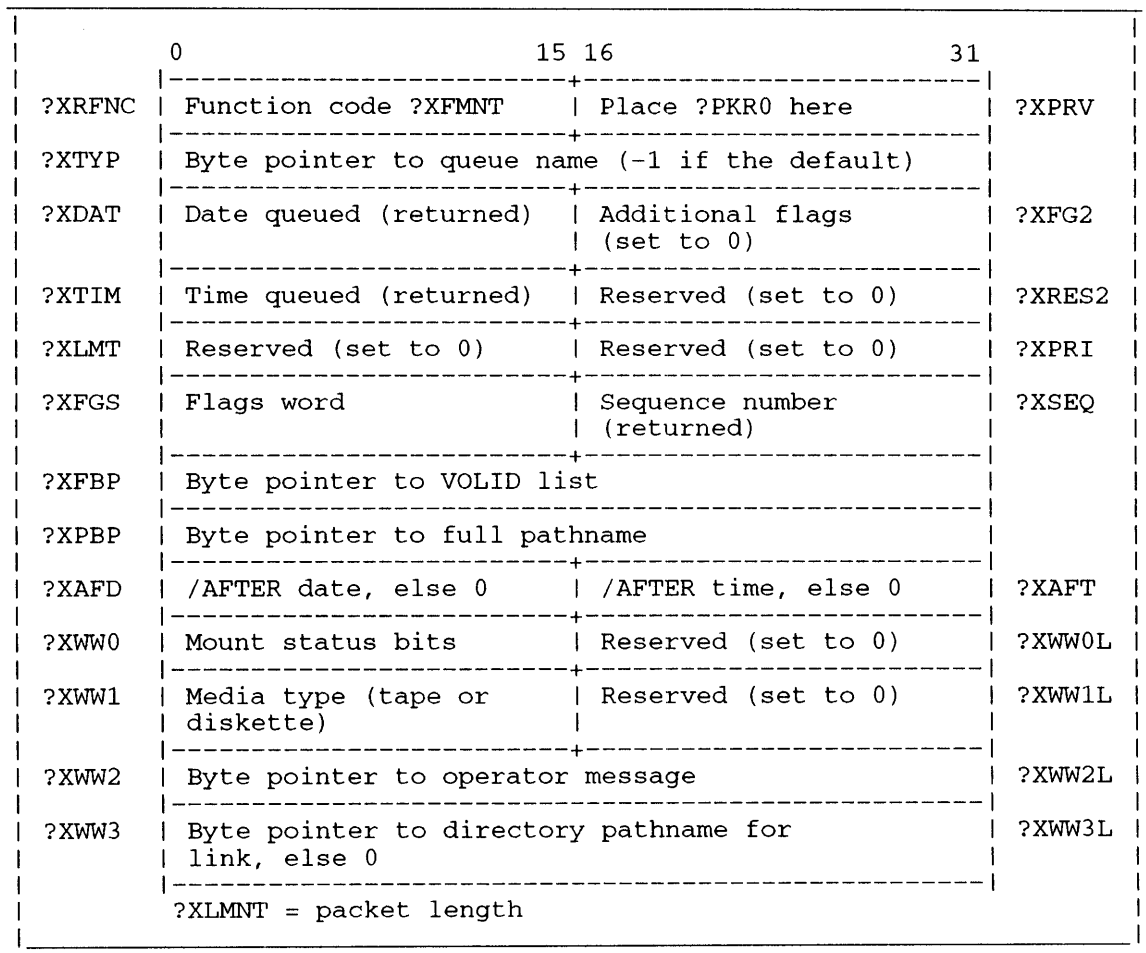


Figure 2–34. Structure of ?EXEC Packet for a MOUNT Queue Request

Dismounting a Unit (Extended Request)

You can specify a directory where the logical name of the dismounted unit will be located. You must also specify this directory pathname at DISMOUNT time.

Issue ?EXEC with the packet shown in Figure 2–35.

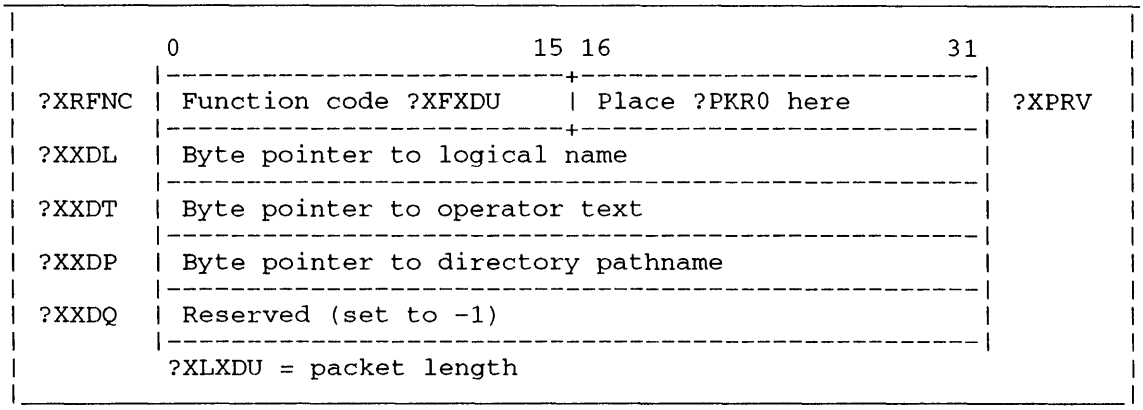


Figure 2–35. Structure of ?EXEC Packet for Dismounting a Unit (extended request)

Modifying the Queuing Parameters of a Queued Job

You use the ?XFMOD function of ?EXEC to request a change in the queuing parameters of a job that you previously queued (except for an ?XFUSR job). The packet in Figure 2–36 shows the structure of a packet for the ?XFMOD function. Offsets whose areas begin with an asterisk (such as ?XFBP) are offsets whose values you can change. You can also reset flags with this version of the ?EXEC call.

Note the similarity of the packet in Figure 2–36 with the submission–type packets such as the one in Figure 2–34.

Table 2–21 gives more information about offsets ?XWM0 through ?XWM3 in Figure 2–36.

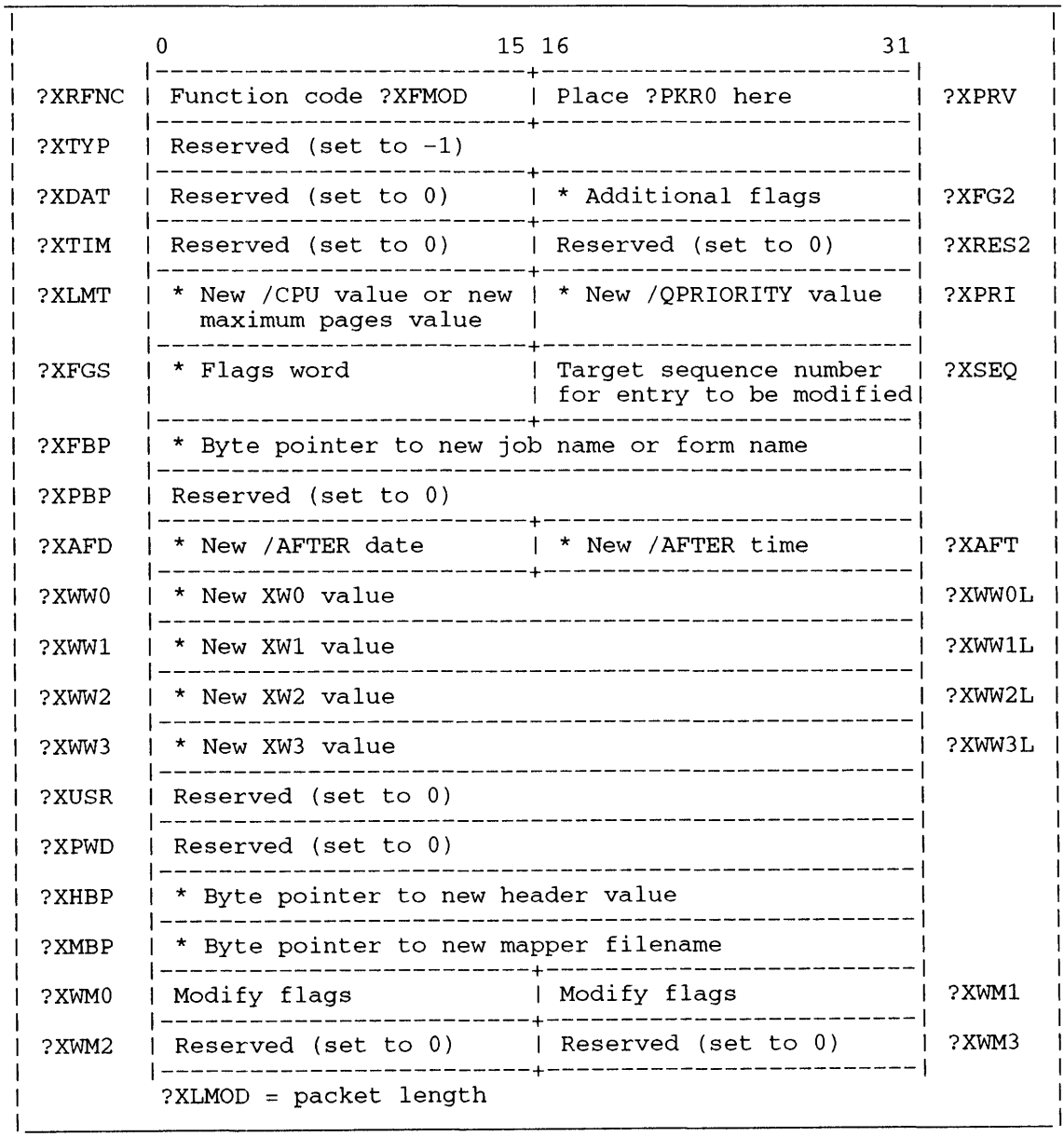


Figure 2–36. Structure of ?EXEC Packet for Changing Queuing Parameters

Required fields in the QMODIFY packet of Figure 2–36 are:

- Offset ?XRFNC must contain function code ?XFMOD.
- Offset ?XPRV must contain value ?PKR0.
- Offset ?XTYP must contain –1 under AOS/VS.
- Offset ?XSEQ must contain the sequence number of the job you want to modify.

Set all unused offsets in the packet to zero. If you want to change any of the original queued parameters, place the new value in the appropriate location in the packet. This location is the same one you would specify if you were queuing the job for the first time.

The bit flags in offsets ?XWM0, ?XWM1, ?XWM2, and ?XWM3 appear in Table 2–21.

Table 2–21. Contents of Selected Offsets in the ?EXEC Packet for the ?XFMOD Function

Offset	Contents
?XWM0	Bit flags, word 0 ---------- Flags 1B0 through 1B3 are Reserved (Set to 0.) When the following flags are set, a corresponding offset contains new information. ?W0LMT = 4 Indicates offset ?XLMT contains a new CPU limit, /CPU ?W0BP = 5 Indicates offset ?XWW0 contains a new beginning page number, /BEGIN ?W0END = 6 Indicates offset ?XWW1 contains a new ending page number, /END ?W0COP = 7 Indicates offset ?XWW2 contains a new number of copies, /COPIES ?W0PG = 8. Indicates offset ?XLMT contains a new pages limit, /PAGES ?W0QPR = 9. Indicates offset ?XPRI contains a new queue priority, /QPRIORITY ?W0JN = 10. Indicates offset ?XFBP contains a byte pointer to a new jobname, /JOBNAME ?W0FBP = 11. Indicates offset ?XFBP contains a byte pointer to a new form name, /FORMS ?W0ADT = 12. Indicates offsets ?XAFD and ?XAFT contain a new /after date and time, /AFTER ?W0QOT = 13. Indicates offset ?XWW0 contains a byte pointer to new queue output file, /QOUTPUT ?W0QLT = 14. Indicates offset ?XWW1 contains a byte pointer to a new queue list file, /QLIST ?W0DES = 15. Indicates offset ?XWW3 contains a byte pointer to a new destination string (for batch, print and plot only), /DEST
?XWM1	Bit flags, word 1 ---------- ?W1SH = 0 Examine bit ?XFSH in word ?XFGS, /HOLD ?W1DA = 1 Examine bit ?XFDA in word ?XFGS, /DELETE ?W1NR = 2 Examine bit ?XFNR in word ?XFGS, /RESTART ?W1BI = 3 Examine bit ?XFBI in word ?XFGS, /BINARY ?W1OP = 4 Examine bit ?XFOP in word ?XFGS, /OPERATOR ?W1NO = 5 Examine bit ?XFNO in word ?XFGS, /NOTIFY ?W1TI = 7 Examine bit ?XFTI in word ?XFGS, /TITLES ?W1FO = 8. Examine bit ?XFFO in word ?XFGS, /FOLDLONGLINES 1B9 through 1B15 -- Reserved (Set to 0.)
?XWM2	Reserved (Set to 0.)
?XWM3	Reserved (Set to 0.)

Obtaining a List of Queue Names

You use the ?XFNQN function of ?EXEC to request a list of queue names. You can issue the call repeatedly to obtain names of all queues of a given type or of all queues regardless of type. Since this call can be iterative, we provide a "get next" key field in the parameter packet. This offset (the 64–bit field formed by ?XFHK and ?XFLK) has an initial value of 0. Successive iterations use and modify this offset. System calls ?GNFN, and ?EXEC with function code ?XFQDS, also use a "get next" key field.

Offset ?XFWP of the main packet contains a word pointer to the first of a series of subpackets. The number of these subpackets is in offset ?XNRQ.

You must create these subpackets. In each subpacket, define its byte pointer. Otherwise, ?EXEC takes the error return with error code ERVBP.

Issue ?EXEC with the packet shown in Figure 2–37.

The ?XQ1FG offset contains the queue status bits shown in Table 2–22.

The flag word in offset ?XFLG contains a bit that indicates whether or not a queue name was provided on input. If so, the byte pointer in the first queue name block points to the queue name. Offset ?XQ1FG contains a bit that indicates the open/closed status of the queue. Offset ?XQQT contains a numeric indicator of the queue type. (See Table 2–23.)

Table 2–22. Queue Status Bit Definitions in Offset ?XQ1FG

Offset	Content
?XQ1FG	Flag word for the queue status bits. The bits are: ?XQSTAT = 0 Open or Closed bit. ?XQMSRT = 1 May or may not start the queue. ?XQDEL = 2 Delete bit. ?XQSRT = 3 Queue started, or did not start. ?XQ1RSV = 4 Reserved (Set to 0.) ?XQ2RSV = 5 Reserved (Set to 0.) Bit Masks for ?XQ1FG word in ?XFNQN block. ?X1STAT = 1B(?XQSTAT) Queue is open ?X1MSRT = 1B(?XQMSRT) May not start the queue ?X1DEL = 1B(?XQDEL) May not delete the queue ?X1SRT = 1B(?XQSRT) Queue is started

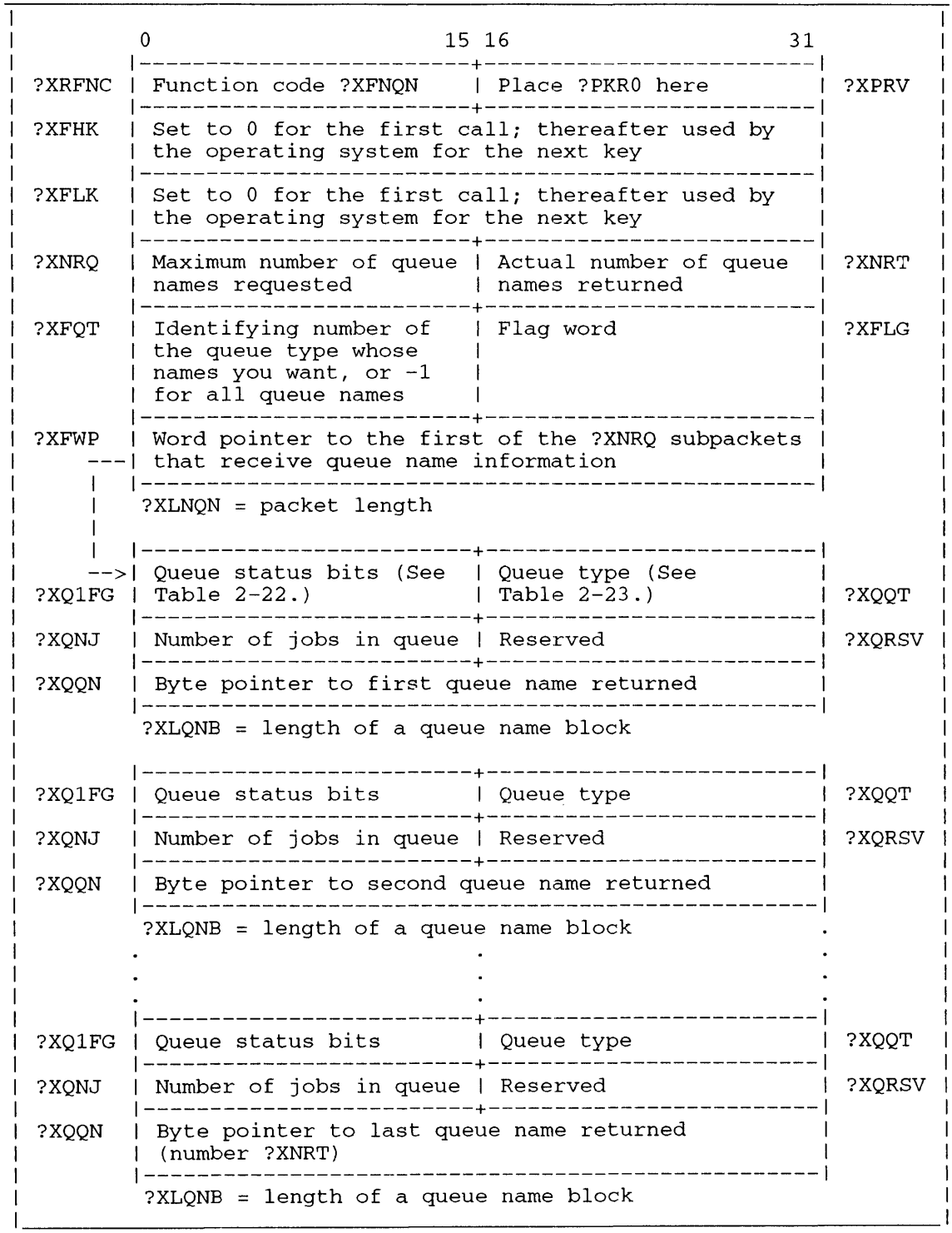


Figure 2–37. Structure of ?EXEC Packet for Obtaining Queue Names

Table 2–23. ?EXEC Queue Types in Offset ?XQQT

Offset Value	Queue Type	Process Type
?XQBAT	BATCH	Batch processing
?XQLPT	PRINT	Print processing
?XQPLT	PLOT	Plot processing
?XQHAM	HAMLET	HAMLET data transfer
?XQSNA	SNA	SNA data transfer
?XQFTA	FTA	FTA data transfer
?XQMUN	MOUNT	Mount processing
?XQUSR	USER	User-defined processing

Obtaining QDISPLAY Information

You use the ?XFQDS function of ?EXEC to obtain QDISPLAY information. Each ?EXEC call of type ?XFQDS can obtain from 1 to 32 units of QDISPLAY information. You specify, in offset ?XFQN, the name of the queue for which you want QDISPLAY information.

Since this call can be iterative, we provide a "get next" key field in the parameter packet. This offset (the 64–bit field formed by ?XFHK and ?XFLK) has an initial value of 0. Successive iterations use and update this offset. System calls ?GNFN and ?EXEC (with function code ?XFNQN) also use a "get next" key field.

To obtain a complete list of all entries in a given queue, issue ?EXEC with key field ?XFHK/?XFLK set to zero. Then repeatedly reissue ?EXEC, without changing fields ?XFHK/?XFLK and ?XFQN, until error END OF GET NEXT SEQUENCE is returned. Repeat this process for each queue for which you need to obtain QDISPLAY information.

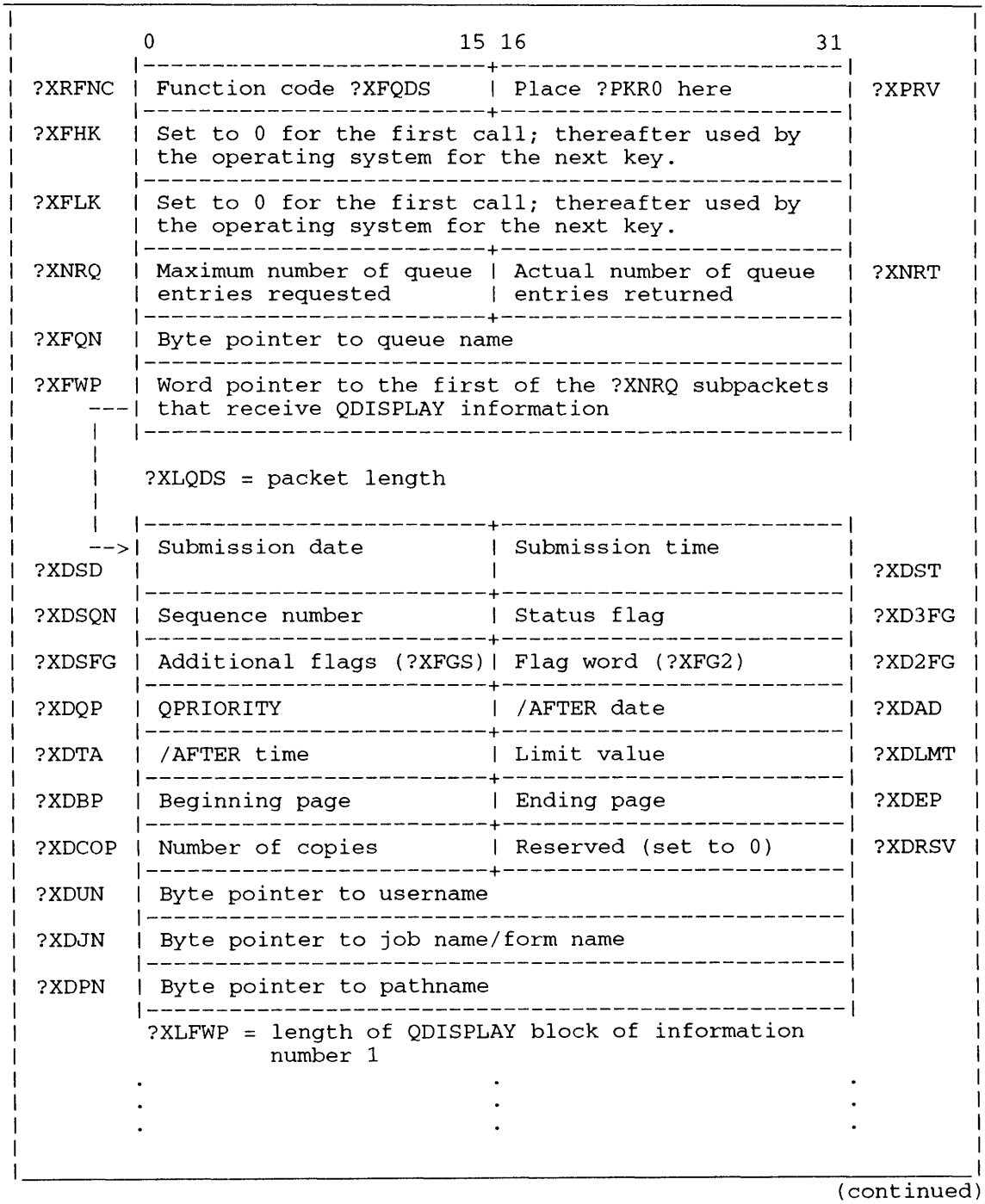
For the flag words in the buffers that receive the information

- The status flag word, offset ?XD3FG, has one bit defined. 0B0 represents an inactive job and 1B0 represents an active job.
- Bit definitions for the flag word, offset ?XD2FG, are the same as for those of ?XFG2 in queue submission packets.
- Bit definitions for the additional flag word, offset ?XDSFG, are the same as for those of ?XFGS in queue submission packets.

Offset ?XFWP of the main packet contains a word pointer to the first of a series of subpackets. The number of these subpackets is in offset ?XNRQ.

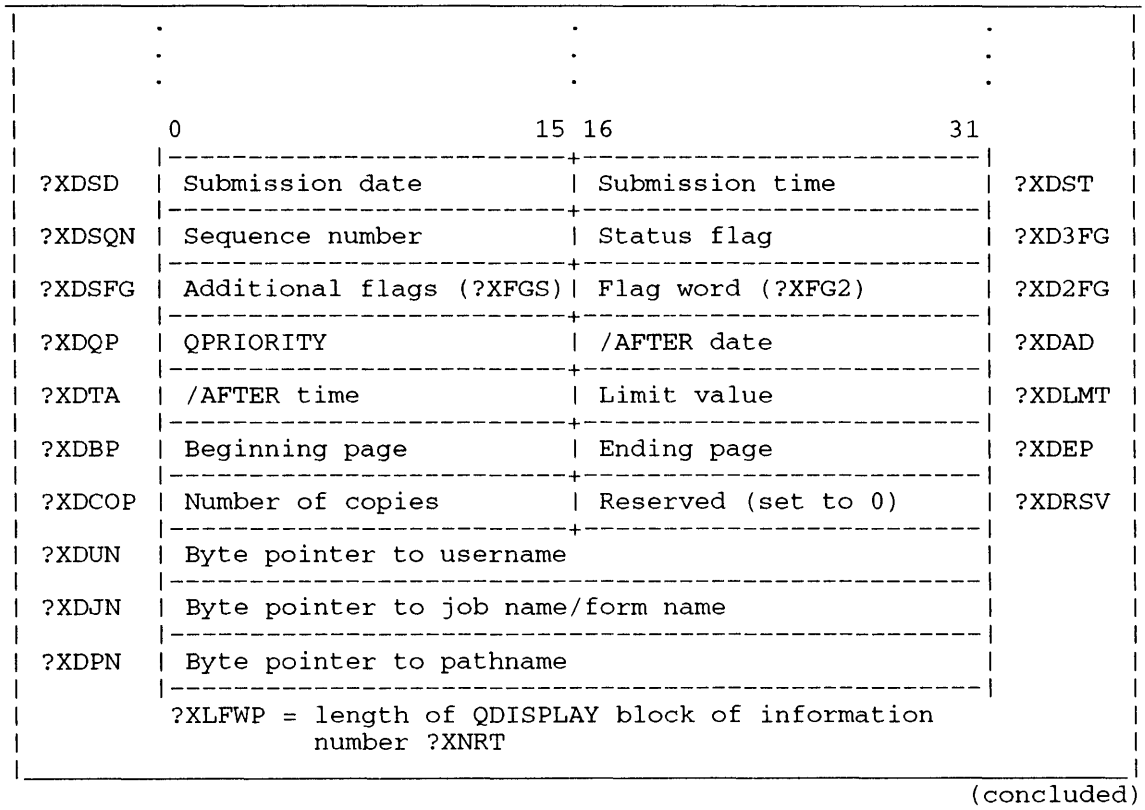
You must create these subpackets. In each subpacket, define its three byte pointers. Otherwise, ?EXEC takes the error return with error code ERVBP.

Issue ?EXEC with the packet shown in Figure 2–38.



(continued)

Figure 2–38. Structure of ?EXEC Packet for Obtaining QDISPLAY Information



(concluded)

Figure 2–38. Structure of ?EXEC Packet for Obtaining QDISPLAY Information

Sample Packet

The following sample packet shows mounting an unlabeled tape:

```
PKT:    .BLK      ?XLMUN              ;Packet length is ?XLMUN.
        .LOC      PKT+?XRFNC          ;What are you going to do?
        .WORD     ?XFMUN              ;Mount an unlabeled tape.
        .LOC      PKT+?XRES           ;Reserved
        .WORD     0                   ;(Set to 0.)
        .LOC      PKT+?XMUL           ;Byte pointer to tape's
                                      ;logical name?
        .DWORD    LNAM*2              ;Byte pointer to LNAM.
        .LOC      PKT+?XMUT           ;Byte pointer to operator
                                      ;message?
        .DWORD    MSG*2               ;Byte pointer to MSG.
        .LOC      PKT+?XLMUN          ;End of packet.
LNAM:   .TXT      /TAPE/              ;Logical tape name.
MSG:    .TXT      /TAPE IS ON FLOOR/  ;Operator text message.
```

Notes

- Refer to the chapter about the EXEC utility in the manual *Managing AOS/VS and AOS/VS II* for information on the EXEC commands, setting queue names, batch processing, and the format directory.

?EXPO Sets, clears, or examines execute–protection status.

?EXPO
error return
normal return

Input

AC0	Starting address of the target region
AC1	Ending address of the target region
AC2	One of the following: • –1 to clear execute protection • 0 to set execute protection • Bit pointer to the start of a bit array that defines the region's execute–protection status

Output

AC0	Unchanged
AC1	Unchanged
AC2	Unchanged

Error Codes in AC0

ERICM	Illegal system command (invalid system call for 16–bit processes)
ERPRE	Invalid system call parameter
ERVWP	Invalid word pointer passed as a system call argument

Why Use It?

You can prevent your program from executing certain logical pages, such as pages that contain data, by setting execute protection. This makes it easier to find errors in your code.

Who Can Use It?

There are no special process privileges needed to issue this call, and there are no restrictions concerning file access.

What It Does

Depending on your input to AC2, ?EXPO performs one of the following functions:

- Sets execute protection for the logical address region that you specify in AC0 and AC1.
- Clears execute protection from the logical address region that you specify in AC1.
- Returns the current execute–protection status for the region that you specify.

Setting and Clearing Execute Protection

To set or clear execute protection for a region, load AC0 with the starting address of the region and load AC1 with the ending address of the region before you issue ?EXPO. The starting and ending addresses must be in the same ring.

When you use ?EXPO to set execute protection, the operating system checks to see if the starting address is the first word on a page and if the ending address is the last word on a page. If either address falls on a page boundary, the operating system includes that page in the execute–protected region. If neither the starting nor the ending address falls on a page boundary, the operating system excludes that page from the execute–protected region.

Examining Execute–Protection Status

To find out whether the specified region is execute protected, perform the following steps:

1. Define a bit array in your address space.

   Bit 0 in this array represents the first page in the target region,

   Bit 1 represents the second page, and so on to the end of the array.

2. Load the array's bit pointer relative to the calling segment into AC2.

3. Issue ?EXPO.

On output, the operating system sets the corresponding bit to 1 if the page is execute protected; it sets the corresponding bit to 0 if the page is not execute protected. In other words, each bit that is set means that the page that that bit represents is execute protected.

?FDAY — Converts date to a scalar value.

?FDAY
error return
normal return

Input

AC0	Days from 1 through 31
AC1	Month from 1 through 12
AC2	(Year minus 1900) from year = 1968 through year = 2145

Output

AC0	Scalar value for days (the number of days that have elapsed since 31 December 1967)
AC1	Unchanged
AC2	Unchanged

Error Codes in AC0

ERPRE	Invalid system call parameter

Why Use It?

?FDAY is a useful way to obtain input parameters for the ?CREATE system call's time block, an additional parameter block you can use to record a file's creation date and time, and the day and time it was last accessed or modified. The parameters for the ?CREATE time block must be scalar values.

Who Can Use It?

There are no special process privileges needed to issue this call, and there are no restrictions concerning file access.

What It Does

?FDAY is the inverse of ?CDAY; that is, it converts the date, expressed in standard notation, to a scalar value from a base date of 31 December 1967. Before you issue ?FDAY, load AC0, AC1, and AC2 with octal values for the day, month, and year, respectively. To obtain the input value for AC2, subtract 1900 from the year.

You can use any legal dates from 1 January 1968 (1,1,68) through 31 December 2145 (31,12,245) as input parameters for ?FDAY. To determine the day of the week from the output, divide the value in AC0 by 7 and note the remainder. Every date with a remainder of 1 (for example, 1 January 1968) fell on Monday.

Notes

- See the description of ?CREATE in this chapter for information on the ?CREATE time block.

?FEDFUNC
error return
normal return

Input

AC0	Unused
AC1	Unused
AC2	Address of packet

Output

AC0	Unchanged
AC1	Unchanged
AC2	Unchanged

Error Codes in AC0

ERUFR	Unknown function request
ERVMP	Invalid address

Why Use It?

?FEDFUNC provides you with a simple–to–use interface to the File Editor (FED) utility.

Who Can Use It?

There are no special process privileges needed to issue this call, and there are no restrictions concerning file access.

What It Does

?FEDFUNC provides you with an interface to some of the FED utility routines.

For example, suppose you want to disassemble an instruction in a program file. You would first issue ?FEDFUNC with its ?FROST function to open the symbol table file of the target program. Next, you would issue ?FEDFUNC with its ?FRDIS function. This issuance would include the instruction to disassemble along with the location (i.e., program counter = PC value) of that instruction in the target program. The result is the disassembled instruction.

Each function takes a unique packet. However, the first offset, which defines the function, is common to all packets. The second offset is also common to all packets.

Here's the correspondence between figure numbers that show the structures of the packets and the functionality of the packets.

Figure 2–39	?FRCR	Change radix
Figure 2–40	?FROST	Open symbol table file
Figure 2–41	?FREFS	Evaluate FED string
Figure 2–42	?FRDIS	Disassemble an instruction
Figure 2–43	?FRITS	Insert a temporary symbol
Figure 2–44	?FRDTS	Delete a temporary symbol

The operating system returns information to the second offset for all function codes except ?FRCR and ?FROST.

Table 2–24 describes the contents of the packet to evaluate a FED string.

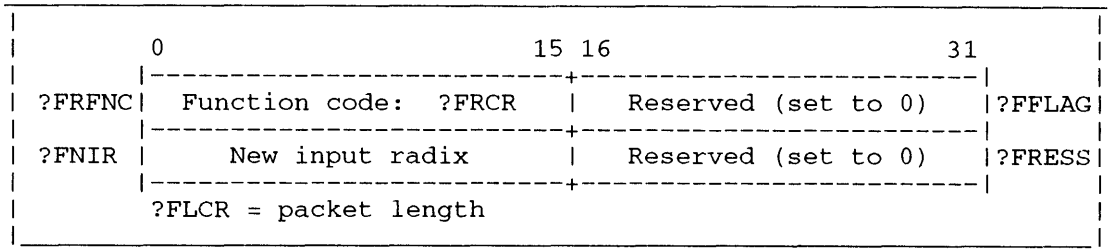


Figure 2–39. Structure of ?FEDFUNC Packet to Change Radix

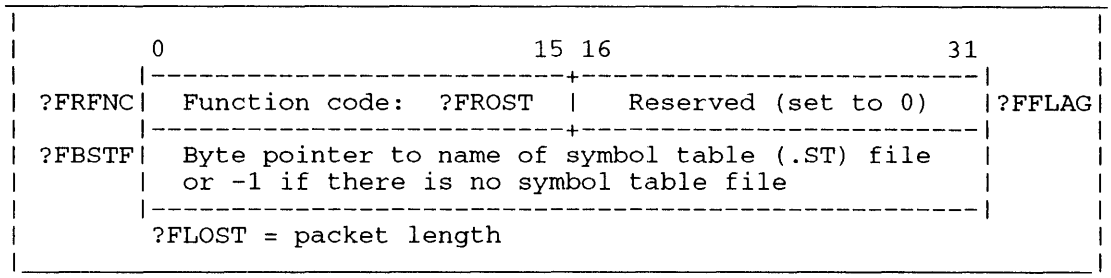


Figure 2–40. Structure of ?FEDFUNC Packet to Open Symbol Table File

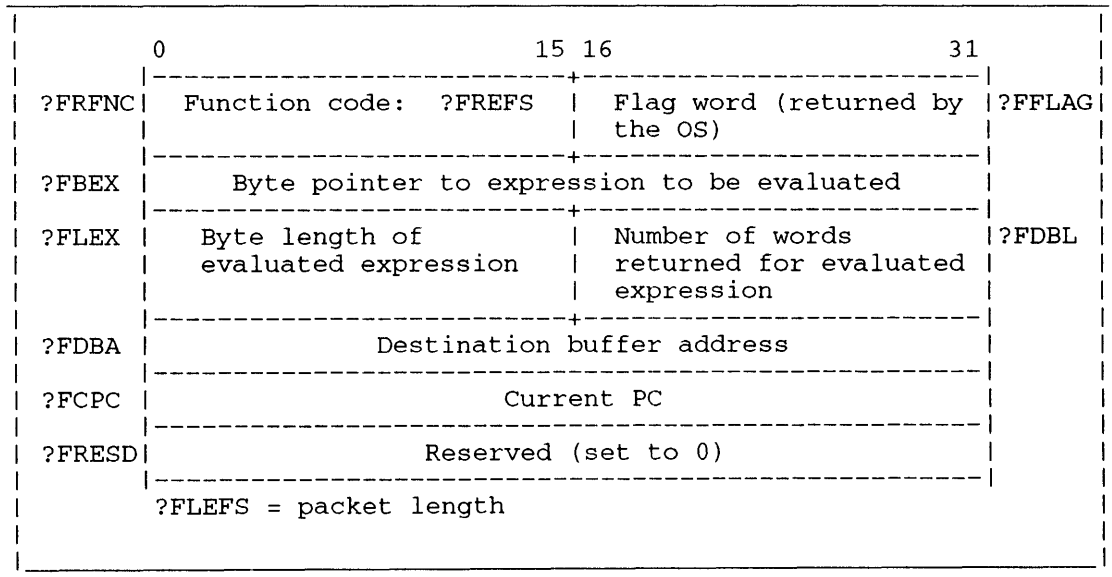


Figure 2–41. Structure of ?FEDFUNC Packet to Evaluate a FED String

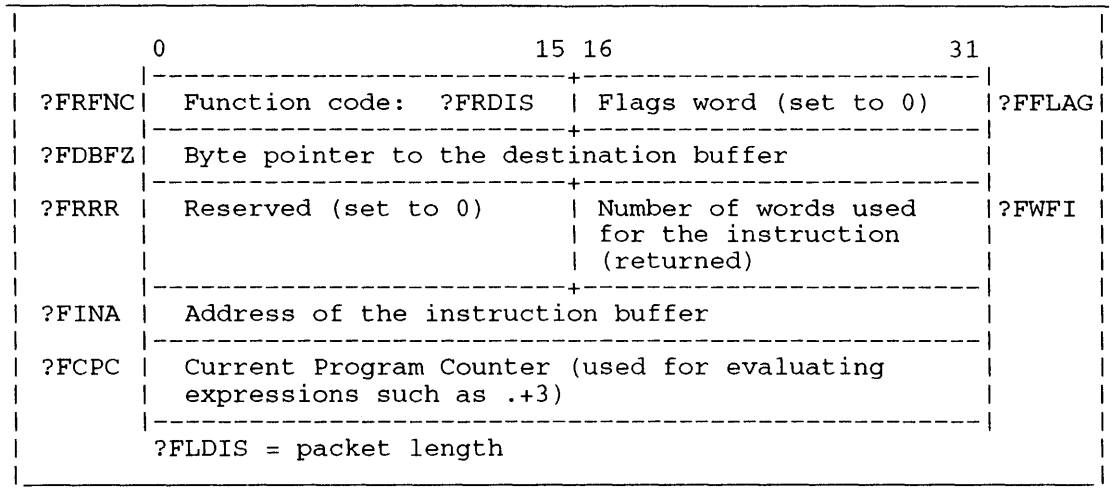


Figure 2–42. Structure of ?FEDFUNC Packet to Disassemble an Instruction

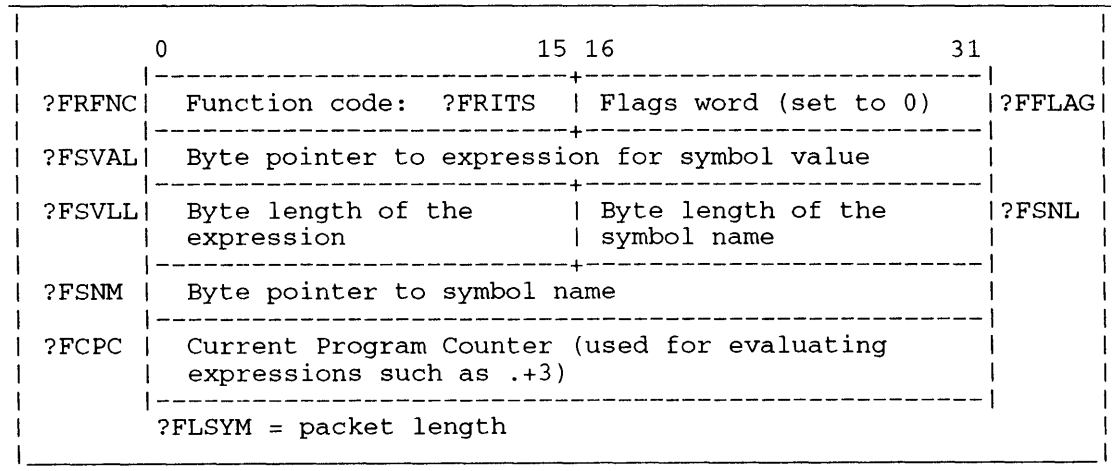


Figure 2–43. Structure of ?FEDFUNC Packet to Insert a Temporary Symbol

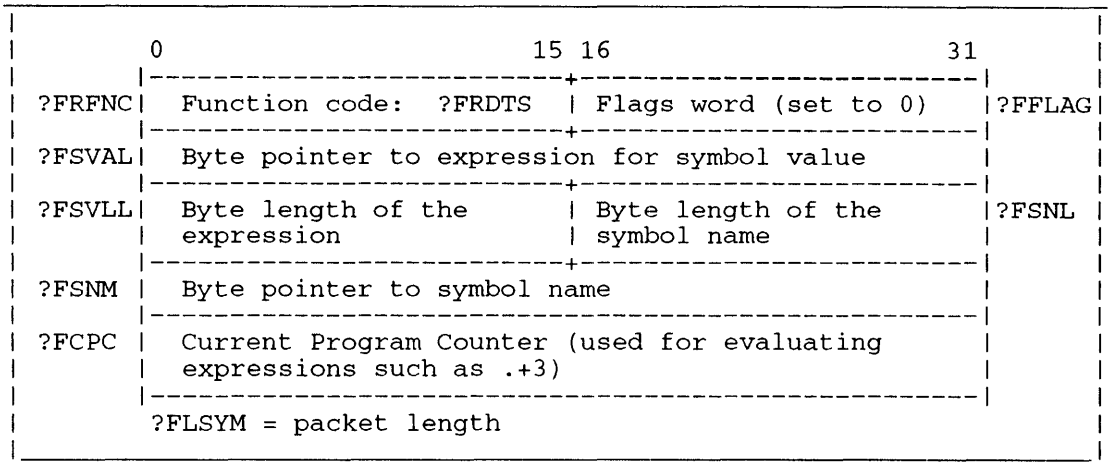


Figure 2–44. Structure of ?FEDFUNC Packet to Delete a Temporary Symbol

Table 2–24. Contents of ?FEDFUNC Packet to Evaluate a FED String*

Offset	Contents
?FRFNC	Contains a code that defines the function that you wish to perform. In this case, the function code is ?FREFS.
?FFLAG	The OS can return the following flag words to this offset: - ?FINST, which means that the string contains an instruction. - ?FFLPT, which means that the string contains a floating-point number. - ?FEXPR, which means that the string contains an expression.
?FBEX (doubleword)	Byte pointer to the expression that you want to evaluate.
?FLEX	The length of the expression that you want to evaluate (in bytes).
?FDBL	The number of flag words that the OS has returned in offset ?FFLAG for the expression that you want to evaluate.
?FDBA (doubleword)	Address of the destination buffer. (The destination buffer that you provide must be at least ?FMEFS bytes long.)
?FCPC (doubleword)	The current PC. (?FEDFUNC uses the current PC to evaluate expressions such as .+3.)
?FRESD (doubleword)	Reserved. (Set to 0.)

* There is no default unless otherwise specified.

?FEOV Forces end–of–volume on labeled magnetic tape.

AOS/VS

?FEOV
error return
normal return

Input

AC0	Reserved (Set to 0.)
AC1	Number of channel on which the magnetic tape was opened
AC2	Reserved (Set to 0.)

Output

AC0	Undefined
AC1	Unchanged
AC2	Undefined

Error Codes in AC0

EREOF	End of file
ERIFT	Illegal file type; file is not a labeled tape

File System Error Codes

Why Use It?

?FEOV allows you to end processing on the current labeled tape volume and swap to the beginning of the next volume.

Who Can Use It?

There are no special process privileges needed to issue this call, and there are no restrictions concerning file access.

What It Does

For output tapes, ?FEOV causes the operating system to flush the current buffer to tape and process the end–of–volume normally. Then, a swap to the next reel occurs.

For input tapes, ?FEOV causes the operating system to space the tape to its logical end. If the operating system detects end–of–volume labels, it swaps to the next volume. However, if the operating system detects end–of–file labels, it returns the error code EREOF in AC0.

AOS/RT32 only

?FIDEF
error return
normal return

Input

AC0 Contains the following:

- Bit 1 is a flag bit:

  Bit 1 = 0 if AC2 specifies the number of data channel map slots needed (no map definition table)

  Bit 1 = 1 if AC2 points to a data channel map definition table

- Bit 2 selects routines:

  Bit 2 = 0 if this device will only execute a user routine

  Bit 2 = 1 if this device will both execute a user routine and drive the Virtual Timer Facility

- Bits 3 through 31 contain the device code for the user–defined device in the range from 1 through 191. (= 277 octal).

AC1 Bits 1 through 31 contain the address of the device's DCT; set Bit 0 to 1 if the device is a data channel (DCH) or burst multiplexor channel (BMC) device

AC2 One of the following:

- Address of the map definition table
- Number of map slots needed (no map definition table) (each map slot accesses 1K words)

Output

AC0 Unchanged

AC1 Unchanged

AC2 Unchanged

?FIDEF Continued

Error Codes in AC0

ERDCH	Data channel map full
ERDNM	Illegal device code (The device code is outside the legal range (1 through 191).)
ERIBS	Device already in use
ERNIX	Interrupt service routine contains no ?IXIT
ERPRE	Invalid system call parameter
ERPRV	Caller is not privileged for this action
ERPTY	Illegal process type

Why Use It?

?FIDEF lets you establish an interface between the operating system and a fast device it does not support when you require low interrupt latency. ?FIDEF and the other user–device system calls are particularly useful if you have applications–specific fast peripheral devices for which you have written special device–driver routines.

Who Can Use It?

The caller must be a resident process and must have privilege ?PVDV to use ?FIDEF. All AOS/RT32 processes have this privilege. There are no restrictions concerning file access.

What It Does

?FIDEF defines a fast user device and its device control table (DCT). The operating system builds an internal DCT based on your DCT specifications, and enters this into its interrupt vector table. The operating system also maps your interrupt service routine into Ring 0.

The device always runs a user routine. In addition, your setting Bit 2 in AC0 results in the device's driving the Virtual Timer Facility (VTF). In this case the operating system calls the VTF driver routine when the user routine issues its ?IXIT system call. Furthermore, the user routine runs first and it normally performs some I/O instructions to reset the interrupting device. The VTF must not be running any timers and it cannot be attached to another interrupt — otherwise, ?FIDEF returns error code ERVUS. When the user routine issues system call ?IRMV or when the process that issued ?FIDEF terminates, the operating system removes the VTF from the user interrupt and the VTF begins running in its default mode (under the real time clock).

It's important to know that the VTF driver routine uses the inverse of the interrupt mask from the DCT to unmask interrupts. Why? Because the driver is reentrant and the interrupt can be nested. So, it's important that you construct the interrupt mask with only the one mask bit set for the particular device. Setting any other bits might result in unintended enabling of interrupts.

Before you issue ?FIDEF, set up a device control table (DCT) in your logical address space and load its address into Bits 1 through 31 of AC1. See Figure 2–45 for the DCT format for issuing ?FIDEF from a 32–bit process — the only process size from which you can issue ?FIDEF. Be sure to set offset ?UDVIS in the DCT to the address of the interrupt service routine for the new device, and define the interrupt service mask in offset ?UDVMS. In addition, you must set offset ?UDVBX (the mailbox) to 0.

For devices that reside on the secondary IOCs (i.e., device codes 64.–191.), you must use PIO instructions to communicate with your device. NOVA I/O is limited to the first IOC.

You must extend the DCT to include both a word pointer to a device termination routine and the size of the interrupt service routine. If you have defined a device termination routine and if your process traps or terminates, control passes to the routine. This transfer of control prevents a runaway ?FIDEF–specified device from altering system databases. Also, the environment would be the same at process termination time as if the device had just requested an interrupt.

To extend the DCT, place ?UDFX in offset ?UDRS. The length of the extended packet is ?UDFE (= ?UDLN+?UDFX) words.

If you issue ?FIDEF against a device that AOS/RT32 is using, you receive error code ERIBS. AOS/RT32 uses devices that were identified to it during the system generation process. An example is an MTD magnetic tape controller. AOS/RT32 also uses the following devices.

Mnemonic	Device Code	Description
BMC	–	Burst Multiplexor Channel
CPU	77	Central Processor
DCH	–	Data Channel
PIT	43	Programmable Interval Timer
RTC	14	Real–Time Clock
SCP	45	System Control Processor
TTI	10	Primary Asynchronous Line Input
TTO	11	Primary Asynchronous Line Output
UPSC	4	Universal Power Supply Controller

In addition, AOS/RT32 uses a DRT (Dual Receive/Transmitter, device code 34) on ECLIPSE MV/1400 DC and MV/2000 DC computers. Another name for this device is DUART.

You can issue ?FIDEF along with ?FIXMT, but you cannot issue ?FIDEF along with ?IXMT.

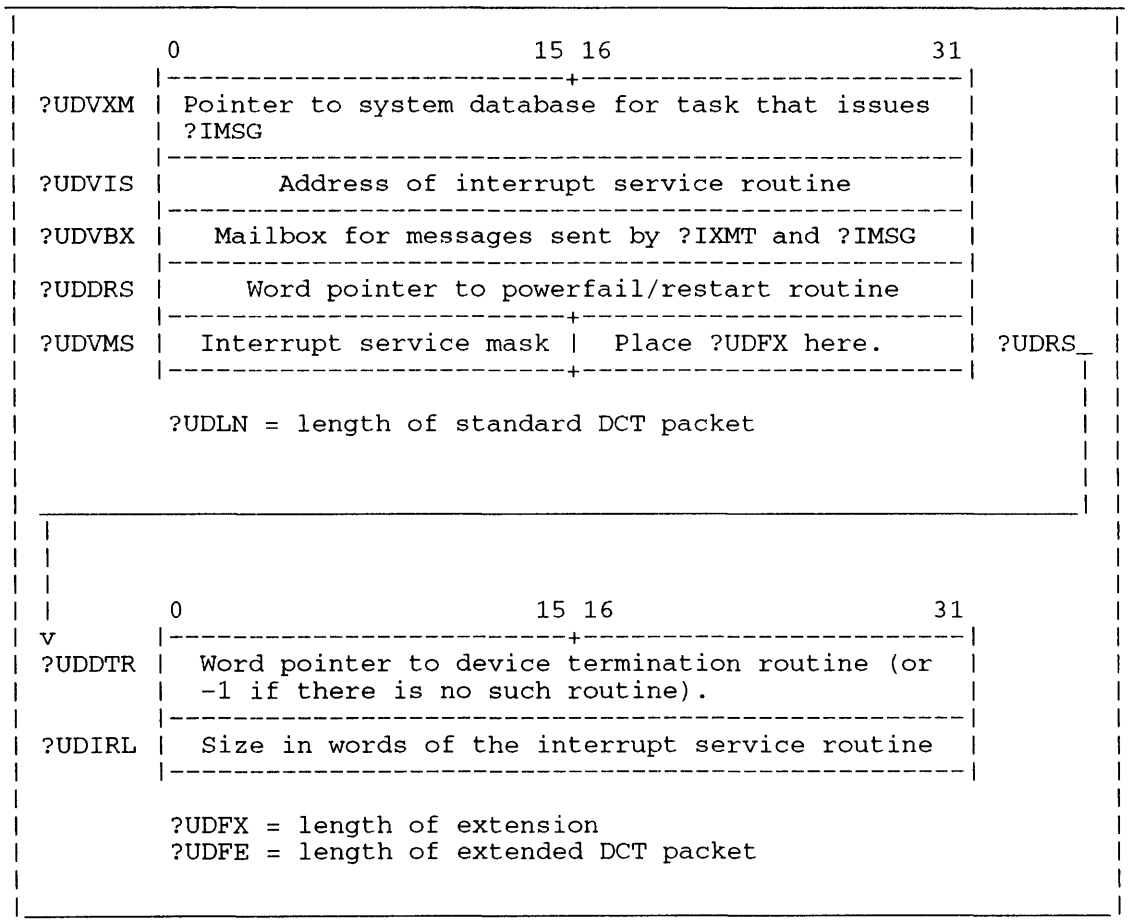


Figure 2–45. Structure of Device Control Table (DCT)

Options

Set Bit 0 of AC1 if you want to use either the data channel (DCH) or the burst multiplexor channel (BMC) for I/O transfers to and from the new device. If you choose the DCH or BMC option, you must also define the number of map slots the device will need. You can load the map slot value into AC2 before you issue ?IDEF or you can set up a map definition table in your address space. If you set up a map definition table, load its address into AC2 before you issue ?IDEF. (If you use AC2, the operating system will allocate map slots in DCH map A.)

The map definition table specifies the first acceptable map slot for BMC or DCH transfers, and optionally, selects a particular DCH map (maps A through P). The map definition table can contain as many as eight entries. Each entry is ?UDELTH words long. The entire table (with eight entries) is ?UDLTH words long. (See Figure 2–46 for the structure of a map definition table entry and see Table 2–25 for a description of its contents.)

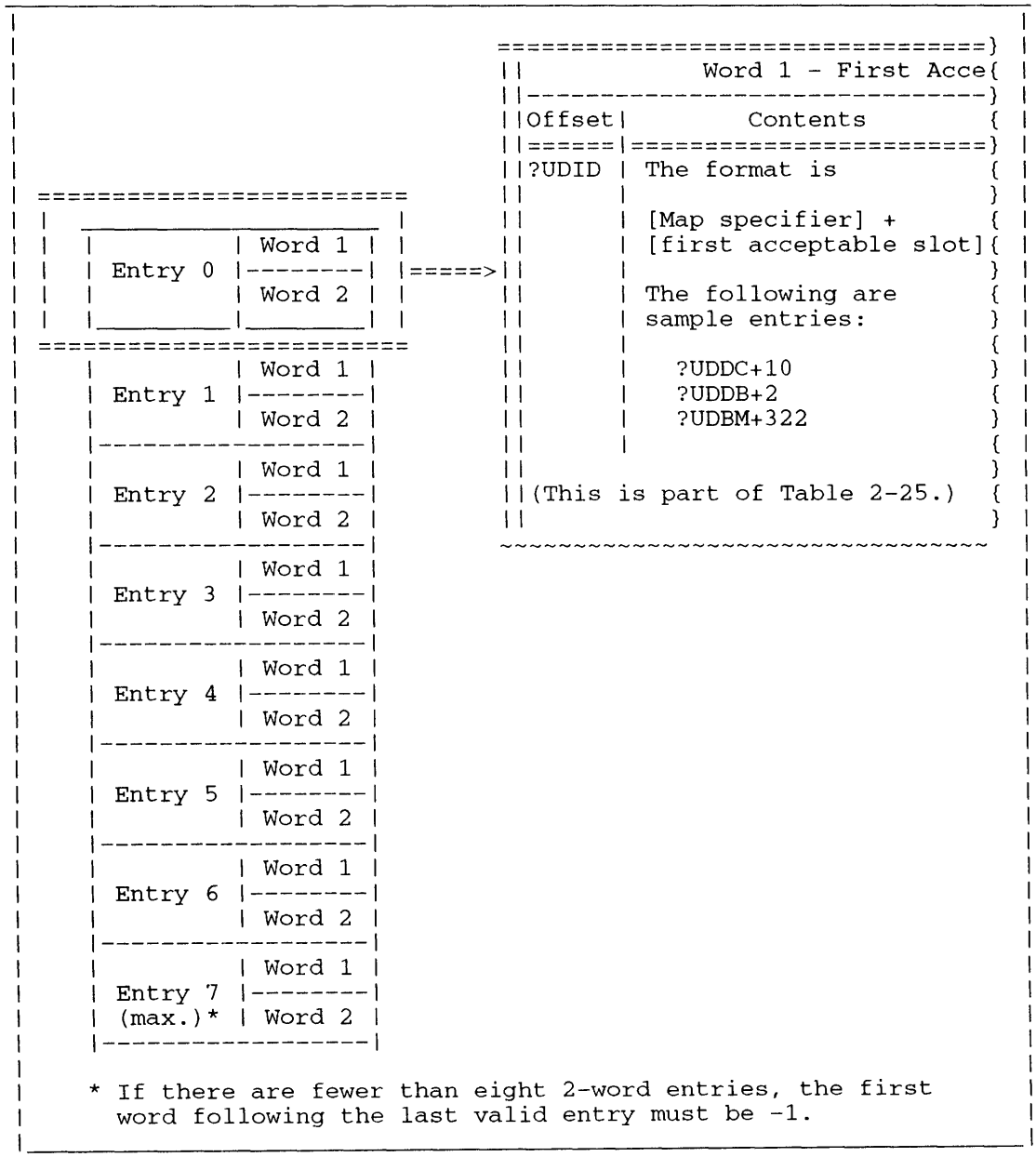


Figure 2–46. Structure of Map Definition Table

Table 2–25. Contents of Map Definition Table Entry

Word 1 - First Acceptable Map Slot		
Offset	Contents	Comments
?UDID	The format is [Map specifier] + [first acceptable slot] The following are sample entries: ?UDDC+10 ?UDDB+2 ?UDBM+322	Map specifier must be one of the following: - ?UDBM, which selects the BMC map. - ?UDDA, which selects the DCH A map. - ?UDDB, which selects the DCH B map. - ?UDDC, which selects the DCH C map. - ?UDDD, which selects the DCH D map. - ... - ?UDDP, which selects the DCH P map. First acceptable slot must be - From 0 through 1777 (octal), if your map specifier is ?UDBM. - From 0 through 37 (octal), if your map specifier is ?UDDA, ?UDDB, ?UDDC, or ?UDDD. The OS allocates the first contiguous group of slots on the map, starting with the first acceptable slot on the map that you selected. Then, the OS returns to you the first slot that it allocated in ?UDID.
Word 2 - Number of Map Slots Requested		
Offset	Contents	Comments
?UDNS	Number of map slots requested in range from 0 through 37 (octal).	The sum of the first acceptable slot plus the number of slots cannot be larger than the size of the map that you requested; that is, 37 (octal) for DCH or 2000 (octal) for BMC.
NOTE: If the OS cannot allocate all entries, then it does not allocate any entries.		

Offset ?UDID, the map identifier word, defines which channel (the BMC or the DCH) the operating system should use for data transfers between the device and memory.

If you select data channel I/O, you can select one of the 16 maps. The symbols ?UDDA through ?UDDP correspond to maps A through P. Add the correct map to the value corresponding to the first acceptable map slot you want the operating system to use. For example, the specification ?UDDA+5 tells the operating system to use map A (?UDDA) and to start with slot 5 in that map, if possible. Offset ?UDNS tells the operating system how many contiguous map slots you will need.

The map slots must be contiguous. Thus, if the first acceptable slot you specify is already in use, the operating system takes the next available map slot with the required number of contiguous slots.

If your devices use data channel maps, you should avoid using the first 128 (A – D) data channel map slots if possible. We advise this because some I/O controllers have restrictions on the number

of data channel maps they can gain access to. Many older I/O controllers use a 17–, 16–, or 15–bit data channel address while newer controllers use a 19–bit data channel address. These numbers make the A through D maps more valuable than the E through P maps. We suggest that you request a data channel map starting at the highest slot the device can gain access to. If the request fails, try the next lower slot and so on. This approach leaves the lower slots available for I/O controllers that are restricted to them.

?FIDEF/?IDEF Differences

You can create initial versions of your programs that issue ?IDEF system calls, debug them, and then replace all occurrences of ?IDEF with ?FIDEF. Next are the differences between ?IDEF and ?FIDEF.

?IDEF — If you define your device via ?IDEF as a standard user device, the interrupt service routine (ISR) remains in Ring 7. When an interrupt occurs control passes through an XVCT instruction to the operating system's first–level interrupt handler, and then to your ISR. In turn, your ISR can communicate with its base–level task indirectly by system calls ?IXMT and ?IMSG or else directly by sharing common address space.

?FIDEF — If you define your device via ?FIDEF as a fast user device, the operating system maps the ISR into Ring 0. The ISR physical pages are thus mapped into two logical address spaces — one in your ring and the other within Ring 0. When an interrupt occurs control passes through an XVCT instruction directly into the mapped ISR within Ring 0. This ISR runs at interrupt level and it is "invisible" to the operating system. The operating system replaces ?FIXMT and ?IXIT intersegment calls by intrasegment calls within Ring 0. During interrupt servicing control remains within Ring 0. The mapped ISR and its base–level task share common address space. Either system calls ?FIXMT and ?IMSG, or using the data area inside the ISR, synchronize and communicate between the moved ISR and its base–level task.

Cautions

The following cautions apply to system call ?FIDEF. *Be sure to follow them strictly!*

- Your ISR must be relocatable; i.e. all ISR code must be relative to the hardware's program counter (PC).
- All ISR code can refer only to a memory area inside the ISR! Invalid memory references within an ISR might cause only a process trap when identified to the ?IDEF–related ISR. These references will cause a fatal system error 110005 from within Ring 0. Similarly, a stack fault you cause within Ring 0 will cause a fatal system error 110001.

?FIDEF Continued

- Don't use privileged instructions inside the ISR! Privileged instructions within your ISR will execute normally because the ISR is located within Ring 0; privileged instructions in an ?IDEF–related ISR will cause a process trap. It's your responsibility to ensure that improper use of privileged instructions will not compromise the system's functioning. If your ISR detects error conditions on which you want to terminate the program or take other remedial action, then you should define a protocol by which the ISR can instruct a base–level task via ?FIXMT to issue ?TERM or otherwise handle the error.

- You must wire all ISR code and data. Not doing this results in a panic whose code is 112040 whenever a page fault occurs at interrupt level and you are trying to refer to instructions or data.

- For short rescheduling times, you must wire the User Runtime Library in Ring 7 (from offsets ?URTB through ?UTSK, inclusive).

- Within ?FIDEF there is no verification for the previous five restrictions, so it is your responsibility to write your ISR correctly.

Sample DCT

```
DCT:     .BLK    ?UDLN           ;Allocate enough space for the standard DCT.
                                 ;  (DCT length = ?UDLN).
         .LOC    DCT+?UDVXM      ;TCB address of ?IMSG task.
         .DWORD  0               ;Set to 0. (The OS supplies this value.)

         .LOC    DCT+?UDVIS      ;Interrupt service routine address.
         .DWORD  PITISR          ;Interrupt service routine address is PITISR.

         .LOC    DCT+?UDVBX      ;?IXMT mailbox.
         .DWORD  0               ;Set to 0.
         .LOC    DCT+?UDDRS      ;Address of power-failure routine.
         .DWORD  -1              ;There is no power-failure routine.
         .LOC    DCT+?UDVMS      ;Interrupt service mask.
         .WORD   1B12+1B13+1B14+1B15
         .LOC    DCT+?UDRS       ;Standard/extended packet indicator.
         .WORD   ?UDFX           ;Set to length of packet extension.
         .LOC    DCT+?UDLN       ;End of standard packet.
DCTEXT:  .BLK    ?UDFX           ;DCT extension.
         .LOC    DCTEXT+?UDDTR   ;Device termination routine pointer.
         .DWORD  -1              ;There is no device termination routine.
         .LOC    DCTEXT+?UDIRL   ;Length of ISR.
         .DWORD  ISREND-ISR
         .LOC    DCT+?UDFE       ;Length of extended packet.
```

Notes

- See the descriptions of ?FIXMT, ?IMSG, ?IRMV and ?IXIT in this chapter.

- See the description of ?FIDEF in this chapter for explanations of user device termination and user device powerfail/restart routines.

?FIXMT — Transmits a message from an interrupt service routine in Ring 0.

AOS/RT32 only

?FIXMT
error return
normal return

Input

AC0	Device code
AC1	Message
AC2	Reserved (Set to 0.)

Output

AC0	Undefined
AC1	Unchanged
AC2	Unchanged

Error Codes in AC0

ERDNM	Illegal device code
ERNOF	?FIDEF did not define the device. Issue ?IXMT instead.
ERPRV	Caller not privileged for this action (function not called from interrupt level)
ERXMT	Mailbox is already in use (that is, the previous message hasn't been processed yet)
ERXMZ	Attempt to transmit illegal message (message = 0)

Why Use It?

?FIXMT and ?IMSG allow you to exchange data between an interrupt service routine (ISR) for a fast device and its base–level task and to synchronize an ISR with its base–level task.

Who Can Use It?

There are no special process privileges needed to issue this call, beyond those that ?FIDEF requires, and there are no restrictions concerning file access. Since you can use ?FIXMT only at interrupt level, you must issue it from a resident process.

What It Does

?FIXMT sends a message up to 32 bits long from an interrupt service routine for a fast device to a specific receiving task outside the sending routine. (The receiving task issues ?IMSG to receive the message.) ?FIXMT, ?IXIT, ?IXMT, and ?SIGNL are the only system calls you can issue from an interrupt service routine.

Before you issue ?FIXMT, load AC0 with the device code associated with the sending routine, and load AC1 with the message. The message must be nonzero, or ?FIXMT fails on error ERXMZ.

When the operating system executes ?FIXMT, it passes the message to the mailbox that it associates with the device. (Be sure to initialize the mailbox before you issue ?FIXMT.) Then, when the receiving task issues the complementary ?IMSG, the operating system passes the message from the mailbox to AC1. If the mailbox already contains a nonzero value, the ?FIXMT fails on error code ERXMT.

?FIXMT Continued

If a sending routine issues ?FIXMT before the receiver issues ?IMSG, the operating system holds the message in the mailbox for later delivery. If the ?IMSG occurs before the ?FIXMT, the operating system suspends the receiving task until the transmission occurs.

You can issue ?FIXMT along with ?FIDEF, but you cannot issue ?FIXMT along with ?IDEF.

Notes

- See the descriptions of ?FIDEF, ?IXIT, and ?IMSG in this chapter.

AOS/VS

?FLOCK *[packet address]*
error return
normal return

Input

AC0	Reserved (Set to 0.)
AC1	Reserved (Set to 0.)
AC2	Address of the ?FLOCK packet, unless you specify the address as an argument to ?FLOCK

Output

AC0	Unchanged
AC1	Unchanged
AC2	Address of the ?FLOCK packet

Error Codes in AC0

ERLCK	Object already locked
ERLNG	Lock not granted — pended request canceled
ERLLE	Lock limit exceeded — more than 1023 locks for one object
ERFNO	Channel not open
ERICN	Illegal channel number
ERVWP	Invalid address passed as system call argument
ER_FS_DIRECTORY_NOT_AVAILABLE	Directory not available because the LDU was force released (AOS/VS II only)

Why Use It?

Use this system call to restrict access to objects. You can use it to prevent cooperating processes from gaining access to the same object at the same time. An object is any entity whose nature requires processes to cooperate so they can properly gain access to it. For example, an object may be a component of a file. On the other hand, an object may be a critical shared code path. Cooperating processes must agree on the meaning of the object numbers.

To achieve the desired cooperation, all processes wanting to gain access to an object must first attempt to lock the object by issuing ?FLOCK. After the operating system grants a lock on an object to a process the process may then gain access to the object. Note that the operating system does not prevent access to a locked object — it's up to the cooperating processes to make sure they don't refer to an object until the operating system has granted them a lock on the object. When a process has completed its access to an object it should release its lock by issuing system call ?FUNLOCK.

?FLOCK offers three types of lock requests: shared, exclusive, and whole file. With shared locks several processes may use an object simultaneously (e.g., reading records), but no process may gain exclusive access (perhaps for writing). Whole file locking can be either shared or exclusive. With whole–file shared locking, only whole–file shared locks are subsequently permitted. With whole–file exclusive locking, no other locks are permitted on the same file or object.

Who Can Use It?

There are no special process privileges needed to issue this call, and there are no restrictions concerning file access.

What It Does

?FLOCK locks an object for use by cooperating processes. An object is locked for EXCLUSIVE or SHARED use based on the type of lock you request. To lock an object, place the number that represents the object in offset ?FLSEL of the parameter packet.

Cooperating processes use a file to represent a group of related objects that the processes want to gain access to. Objects may represent components (such as records) of the file, but this is not necessary. The file can be empty and the objects may represent entities that are separate from the file. Your process must open the file before it can lock one of its objects. Place the channel number that the open request (i.e., the ?OPEN system call) returned, along with the type of lock, in the ?FLOCK parameter packet.

Cooperating processes must have access to the same file so they can use the locking mechanism that ?FLOCK provides. Note that the operating system does *not* prevent access to a locked object — it is the responsibility of the cooperating processes not to attempt access to an object until the operating system grants them a lock on the object. You can prevent noncooperating processes from gaining access to the objects by setting the file's access control list accordingly.

For more complex situations, we suggest using a SHARED PROTECTED file (see ?SOPPF). A global server process that first opens the file controls access to a shared protected file. To prevent unauthorized access to the file when the global server process is not running, set the file's access control list to null. The global server process must then turn on its Superuser privilege so it can open the file.

To gain access to objects that the file represents, cooperating processes can issue ?RINGLD against a local server routine that then becomes a customer (see ?CON) of the global server. Upon request from a local server the global server can grant permission (see ?PMTPF) for the local server to become a subsequent opener of the file. This permission supersedes the file's access control list. The local server can then issue ?SOPPF against the file, and next use the file to represent objects by issuing ?FLOCK on the channel number that ?SOPPF returns. If these objects represent components of the file's contents, the local server can read the file by issuing ?SPAGE to gain access to an object.

Figure 2–47 shows the structure of ?FLOCK's parameter packet, and Table 2–26 describes its contents.

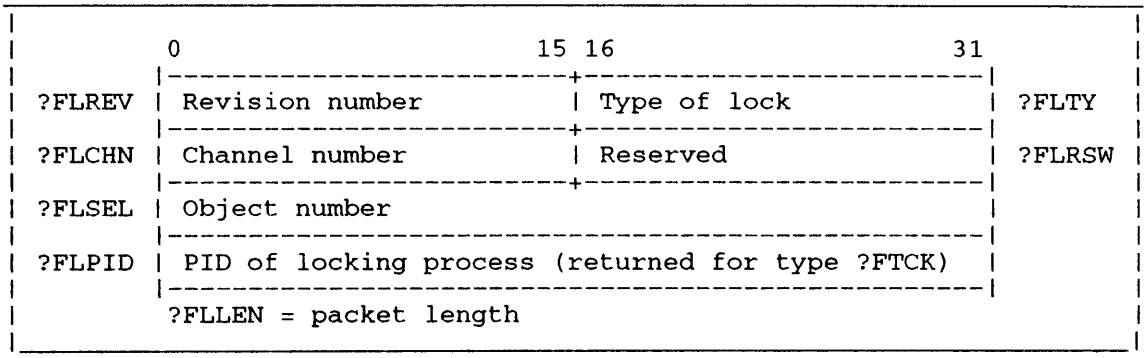


Figure 2–47. Structure of ?FLOCK Packet

086–000195 updates
093–000542

Table 2–26. Contents of ?FLOCK Packet*

Offset	Contents
?FLREV	Packet revision number. Place ?PKR0 here.
?FLTY	Type of lock. It specifies the operation you want. Select from the following values. ?FTCK -- Check for a lock. If selected, AOS/VS does no locking but it returns the status of the object specified. If it is unlocked, then ?FLTY contains 0; if it is locked, then ?FLTY contains the lock type (?FTSH, ?FTEX, or ?FWFL). ?FTEX -- Create an exclusive lock. ?FTSH -- Create a shared lock. ?FWFL -- Lock the whole file. If you lock the whole file, you must also set either ?FTEX or ?FTSH. If you supplied ?FTEX, ?FTSH, or ?FWFL, then you should also specify one of the following pending actions. ?FTER -- Take an error return if AOS/VS is unable to lock the file element. ?FTPN -- Pend if not able to lock, and wait for an unlock to occur. AOS/VS doesn't return an error.
?FLCHN	Supply the channel number for the file.
?FLRSW	Reserved. (Set to 0.)
?FLSEL (double-word)	Object number. Your process and cooperating processes must agree on the meaning of this number.
?FLPID (double-word)	PID of the locking process. If the value of ?FTCK is set in offset ?FLTY, AOS/VS returns the PID. Otherwise, supply 0.

* There is no default unless otherwise specified.

?FLUSH — Flushes the contents of a shared page to disk.

?FLUSH
error return
normal return

Input

AC0	Any address in the page you want to flush to disk (The OS derives the logical page number by stripping off Bits 22 through 31.)
AC1	Reserved (Set to 0.)
AC2	Reserved (Set to 0.)

Output

AC0	Unchanged
AC1	Undefined
AC2	Undefined

Error Codes in AC0

ERNSA Shared I/O request not to shared area
ER_FS_DIRECTORY_NOT_AVAILABLE
Directory not available because the LDU was force released (AOS/VS II only)

Why Use It?

?FLUSH directs the operating system to write a shared page out to disk immediately. Therefore, ?FLUSH is useful if you have modified a shared page and you want to make sure that the operating system updates it immediately.

Note that before you can use ?FLUSH, you must use ?SOPEN to open the target file for shared access.

Who Can Use It?

There are no special process privileges needed to issue this call, and there are no restrictions concerning file access.

What It Does

?FLUSH copies the contents of a shared page from memory to disk, without actually releasing the shared page from memory. The calling process regains control after the operating system has performed the flush.

Notes

- See the description of ?RPAGE and ?ESFF in this chapter.
- See the descriptions of ?SOPEN and ?UPDATE in this chapter for information on simulating ?FLUSH.

?FSTAT *[file status packet address]*
error return
normal return

Input

AC0	may contain one of the following: • Byte pointer to the pathname of the target file • Channel number of the target file
AC1	Flags: Bit 0 = 0 if AC0 contains a byte pointer Bit 0 = 1 if AC0 contains a channel number Bit 1 = 0 to resolve links in the pathname Bit 1 = 1 to ignore links in the pathname
AC2	Address of the file status packet, unless you specify the address as an argument to ?FSTAT

Output

AC0	Unchanged
AC1	Unchanged
AC2	Address of file status packet

Error Codes in AC0

ERCIU	Channel in use
ERFAD	File access denied
ERFNO	Channel not open
ERICN	Illegal channel
ERMPR	System call parameter address error
ERVBP	Invalid byte pointer passed as a system call argument
ERVWP	Invalid word pointer passed as a system call argument
ER_FS_DIRECTORY_NOT_AVAILABLE	Directory not available because the LDU was force released (AOS/VS II only)

Why Use It?

?FSTAT lets you determine the specifications set for a file or a directory when it was created. ?FSTAT also returns the date and time the file was last accessed or modified. (You may need this information before you open or reopen a file.)

Who Can Use It?

There are no special process privileges needed to issue this call. See the following section "What It Does" for an explanation of file access requirements.

What It Does

?FSTAT returns the status parameters of the target file. You can specify the packet address as an argument to ?FSTAT or you can load the address into AC2 before you issue ?FSTAT. Before the calling process can issue ?FSTAT, it must meet these rules:

- If it has no access rights to a file, it needs Read and Execute access to the file's directory.
- If it has any access right to a file, it needs Execute access to the file's directory.

Before you issue ?FSTAT, load AC0 with either the channel number of the target file or a byte pointer to its name.

If you specify a pathname that ends in a link and Bit 1 of AC1 equals 1, the operating system returns status information about the link without resolving it. If the pathname ends in a link and you do not set Bit 1 of AC1, the operating system resolves the link entry and returns status information about the file to which the link refers.

The format of the information ?FSTAT returns varies, depending on the type of file. The operating system returns different information packets for IPC files, directories, unit files, and other file types. Unit files are devices that have been opened as a unit (for example, MCAs, MTBs, disks, etc.). Figure 2–48, Figure 2–49, Figure 2–50, and Figure 2–51 show the structure of the various ?FSTAT packets.

Unit File Packet

In the packet for unit files, offset ?SDCU contains the device address and unit number of the peripheral. (See Figure 2–48 and PARU.32.SR.) The device address contains an I/O Controller (IOC) number (bits 16 and 17) and a 6–bit device code (bits 18 through 23). If your system has more than four I/O Controllers or you request a status on a MRC–connected device, ?FSTAT returns a zero in offset ?SDCU. For AOS/VS II systems with more than four I/O Controllers or a MRC bus, issue the ?XFSTAT system call.

For AOS/VS, offset ?STCH and ?STCL specify the date and time the file was created. For AOS/VS II, offset ?STCH through ?STCL specify the date and time the file was created, last accessed, and modified. These offsets contain the same information as the offsets in the time block for the ?CREATE system call.

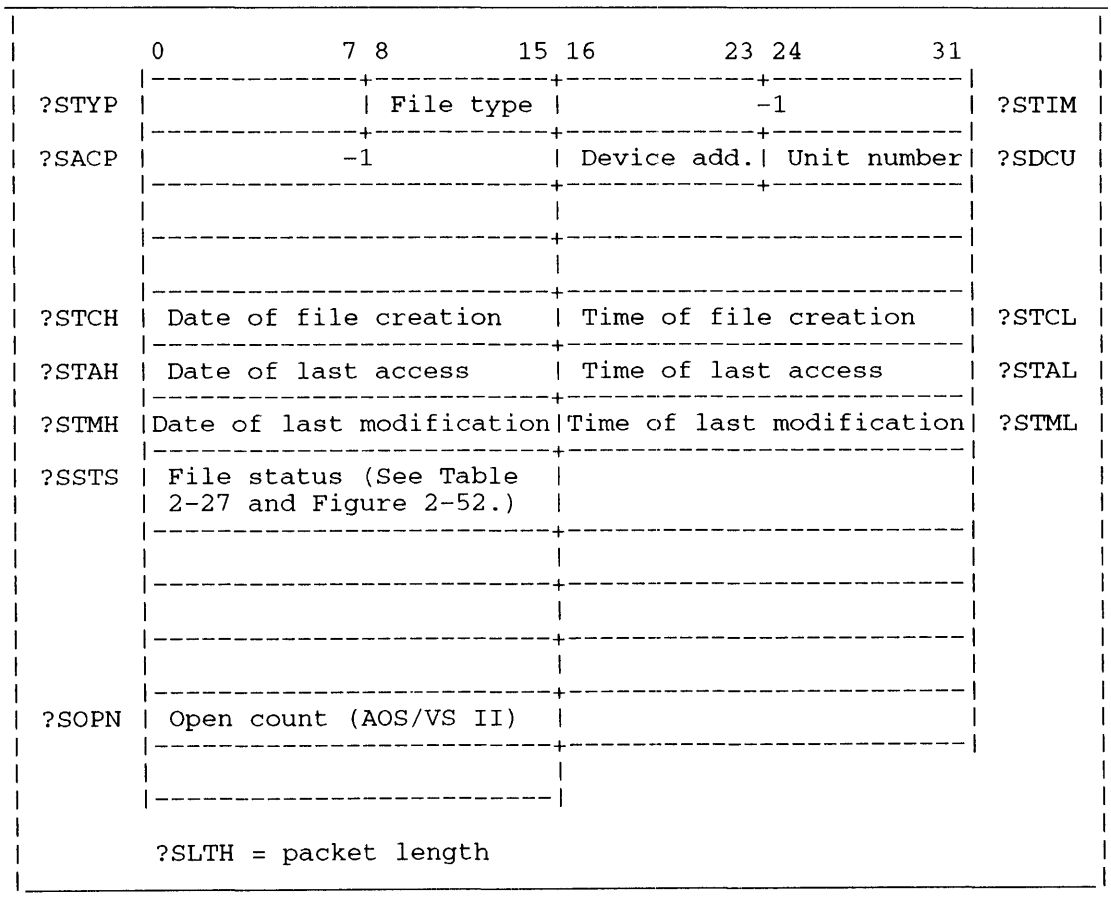


Figure 2–48. Structure of ?FSTAT Packet for Unit Files

IPC File Packet

In the packet for IPC files, offsets ?SPNH and ?SPNL contain the global port number of the IPC file.

For AOS/VS, offset ?STCH and ?STCL specify the date and time the file was created. For AOS/VS II, offset ?STCH through ?STCL specify the date and time the file was created, last accessed, and modified. These offsets contain the same information as the offsets in the time block for the ?CREATE system call.

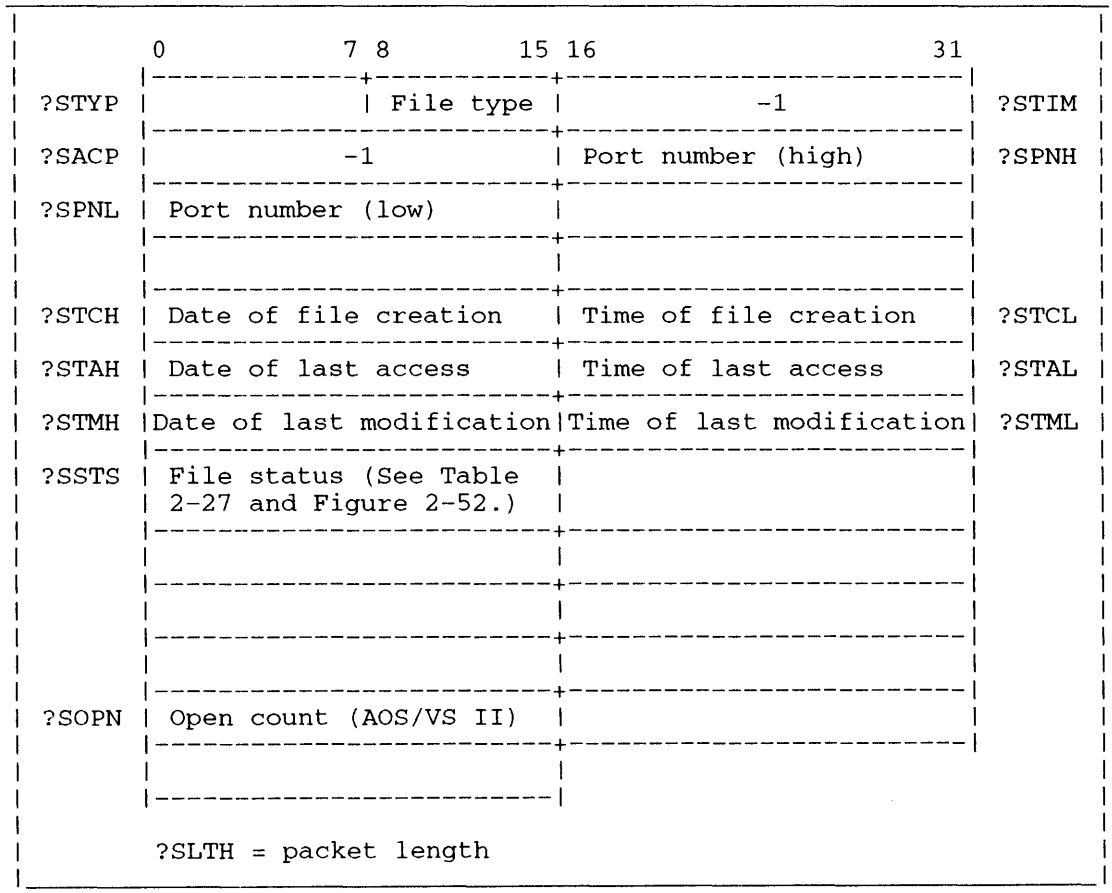


Figure 2–49. Structure of ?FSTAT Packet for IPC Files

Directory/Remaining Types Packets

If the target entry is an AOS/VS directory (DIR or CPD), the operating system returns the hashframe size of the directory in the right byte of offset ?SHFS. (See Figure 2–50.) For AOS/VS II, the operating system returns a zero in offset ?SHFS. (The AOS/VS II file system calculates the hashframe size — no longer a user input.)

The operating system returns in ?SMSH the maximum number of disk blocks available (MS value) for a control point directory (CPD) or a logical disk unit (LDU).

For AOS/VS, the operating system returns in ?SCSH the current number of disk blocks in use, including the inferior file space, for a control point directory. For AOS/VS II, the operating system returns in ?SCSH the current number of disk blocks in use, including the inferior file space, for all directory file types.

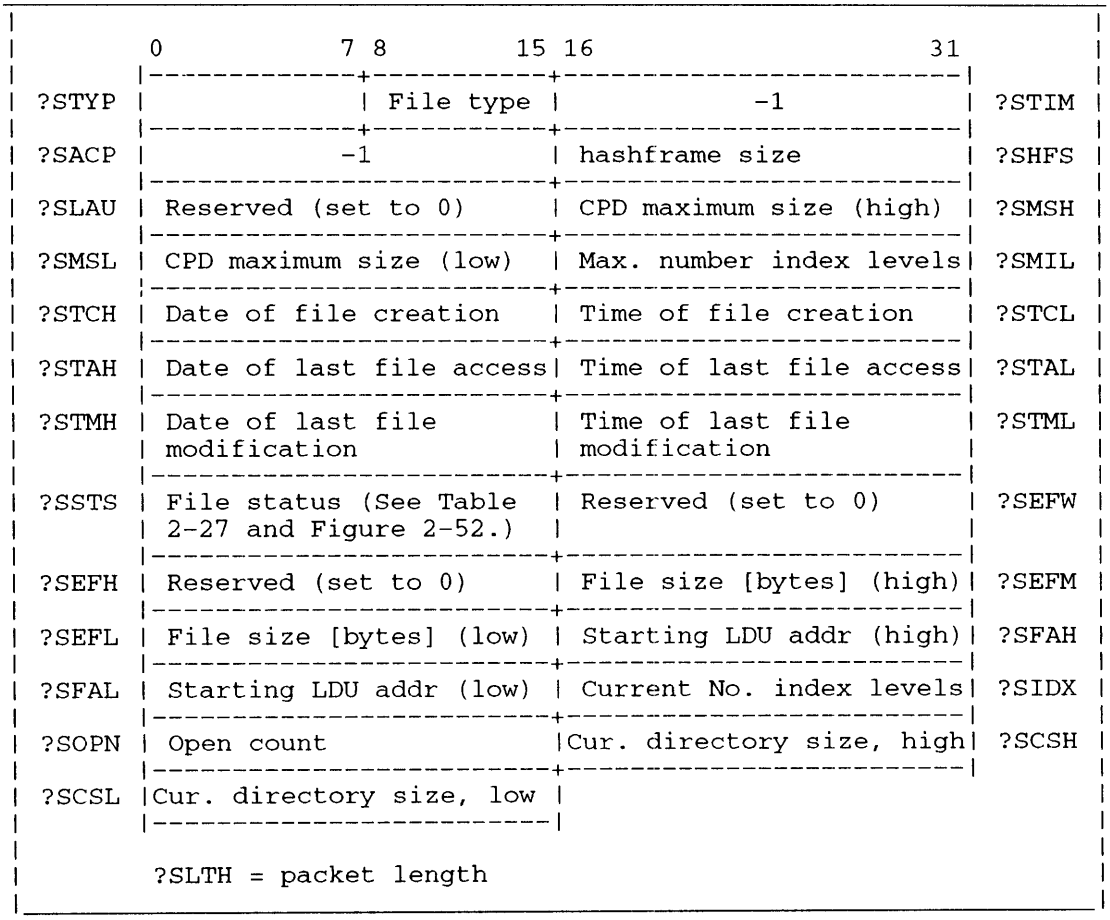


Figure 2–50. Structure of ?FSTAT Packet for Directory Files

Offset ?SMIL indicates the maximum number of index levels allowed for both standard and control point directories.

In an AOS/VS packet for the other file types, offset ?SDEH specifies the element size of the target file. In an AOS/VS II packet for the other file types, offset ?SDEH corresponds to the Primary Element Size defined in ?XCREATE.

Offset ?SMIL specifies the maximum index level, and ?SCPS specifies the file control parameters. (See Figure 2–51.) Offset ?SCPS, which applies only to files with fixed–length records, equals the file's maximum record length. The contents of ?SCPS are undefined for files with other record types. The contents of ?SCPS are the same as the contents of offset ?CCPS in the ?CREATE packet and the contents of offset ?OPFC in the ?GOPEN packet. Offset ?SCPS equals the Host ID for network type files (which is when the right byte of offset ?STYP contains ?REM). Offsets ?SCSH and ?SCSL return the number of blocks currently allocated to the AOS/VS II file.

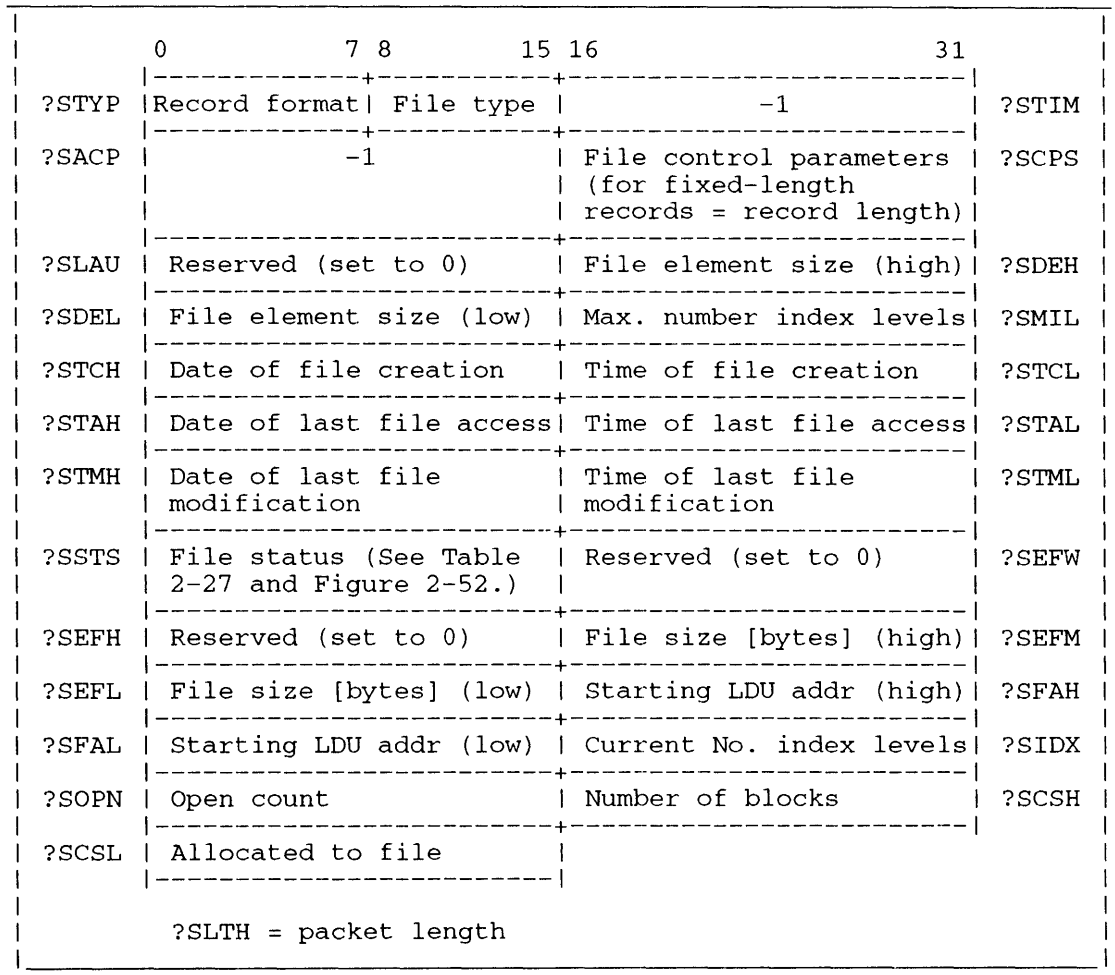


Figure 2–51. Structure of ?FSTAT Packet for Other File Types

Offsets ?STCH through ?STML specify the time the file was created, the time it was last accessed, and the time it was last modified. These offsets correspond to the parameters in the ?CREATE system call's time block.

The operating system describes certain characteristics of the file by returning one or more flag bits to ?SSTS, the file status word. See Table 2–27 for a complete list of these flags.

Table 2–27. Flags Returned in Offset ?SSTS

Flag	Meaning
?FSHB	The file is a shared file.
?FMDB	The file was modified.
?FPRM	The file or directory has the Permanence attribute.
?FDLE	The OS deletes the file after the last ?CLOSE.
?FUDA	The file has an associated UDA (user data area).
?FARA	All users have read access to the file or directory.
?FAEA	All users have execute access to the file or directory.

Figure 2–52 shows the structure of offset ?SSTS.

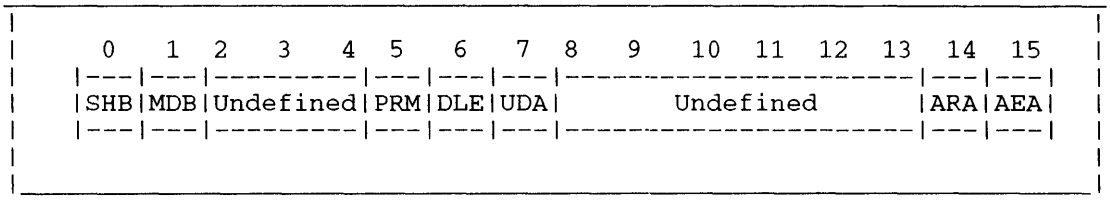


Figure 2–52. ?SSTS Structure

Offsets ?SEFM/?SEFL specify the file's byte length.

Offset ?SFAH specifies the file's starting address in the logical disk. This is a double–precision integer, and can range from 0 to the highest block on the logical disk minus 1.

The operating system returns offset ?SIDX as the current number of index levels (indexes) for the file. This value should not exceed the value of ?SMIL, the maximum number of index levels allowed to the file.

The open count, offset ?SOPN, records the number of users currently using the file for I/O (the number of ?OPEN, ?GOPEN, etc. calls without comparable ?CLOSE, ?GCLOSE, etc. calls). AOS/VS returns offset ?SOPN for the Directory and Other file types. AOS/VS II returns offset ?SOPN for all of the file types.

You can only retrieve the length of a file via ?FSTAT after you have opened, reopened, or just created the file. In other words, you cannot retrieve the length correctly while you are currently writing to the file.

Sample Packet

The following is a sample of an ?FSTAT IPC file packet:

```
PKT:    .BLK    ?SLTH           ;Allocate enough space for the packet
                                ;(packet length = ?SLTH).
        .LOC    PKT+?STYP       ;File type.
        .WORD   0               ;File type for IPC files returned.
        .LOC    PKT+?SPNH       ;Port number.
        .DWORD  0               ;Port number returned.
        .LOC    PKT+?STCH       ;File creation date.
        .WORD   0               ;File creation date returned.
        .LOC    PKT+?STCL       ;File creation time.
        .WORD   0               ;File creation time returned.
        .LOC    PKT+?SSTS       ;Reserved.
        .WORD   0               ;You must set this value to 0.
        .LOC    PKT+?SLTH       ;End of packet.
```

Notes

- See the description of ?RECREATE in this chapter.
- For AOS/VS II, see the description of the ?XCREATE and ?XFSTAT system calls for the additional file structuring parameters that the ?XCREATE system call sets and the ?XFSTAT call displays.

?FTOD — Converts time of day to a scalar value.

?FTOD
error return
normal return

Input

AC0	Seconds from 0 through 59
AC1	Minutes from 0 through 59
AC2	Hour from 0 (midnight) through 23 (11 p.m.)

Output

AC0	Number of biseconds (seconds/2) since midnight
AC1	Unchanged
AC2	Unchanged

Error Codes in AC0

ERPRE	Invalid system call parameter

Why Use It?

You can use ?FTOD to obtain input parameters for the ?CREATE system call's time block.

Who Can Use It?

There are no special process privileges needed to issue this call, and there are no restrictions concerning file access.

What It Does

?FTOD takes time in seconds, minutes, and hours and converts it to a scalar value; that is, the number of biseconds (half the number of seconds) since midnight.

Before you issue ?FTOD, load AC0, AC1, and AC2 with the appropriate octal values for seconds, minutes, and hours, respectively. The legal input parameters for ?FTOD range from 0:00:00 (midnight) to 23:59:59 (11:59 p.m.).

The operating system returns the scalar time value to AC0. If the scalar conversion results in an odd number of seconds, the operating system rounds the value to the next number before division; for example, 7 seconds would result in (8/2) = 4 biseconds.

AOS/VS

?FUNLOCK *[packet address]*
error return
normal return

Input

AC0	Reserved (Set to 0.)
AC1	Reserved (Set to 0.)
AC2	Address of the ?FUNLOCK packet, unless you specify the address as an argument to ?FUNLOCK

Output

AC0	Unchanged
AC1	Unchanged
AC2	Address of the ?FUNLOCK packet

Error Codes in AC0

ERNLK Object not locked
ERVWP Invalid address passed as system call argument
ER_FS_DIRECTORY_NOT_AVAILABLE
Directory not available because the LDU was force released (AOS/VS II only)

Why Use It?

Use this system call to unlock an object when you no longer need it.

Who Can Use It?

There are no special process privileges needed to issue this call, and there are no restrictions concerning file access.

What It Does

?FUNLOCK releases a lock that your process has placed.

?FUNLOCK lets other processes gain a lock on the same object. It can also unlock all locks that your process holds on the specified file. You must have issued ?OPEN for a file before you can unlock one of its objects. Place the channel number ?OPEN returns in the ?FUNLOCK packet.

Cooperating processes must have access to the same file in order to use the unlocking mechanism that ?FUNLOCK provides. The processes must also agree on the meaning of the object numbers so they can guarantee successful unlocking.

Figure 2–53 shows the structure of ?FUNLOCK's parameter packet, and Table 2–28 describes its contents.

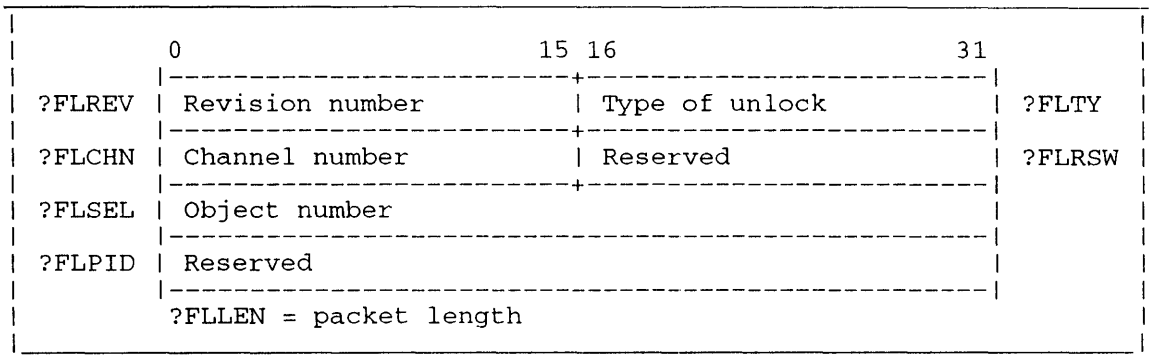


Figure 2–53. Structure of ?FUNLOCK Packet

Table 2–28. Contents of ?FUNLOCK Packet*

Offset	Contents
?FLREV	Packet revision number. Place ?PKR0 here.
?FLTY	Type of unlock. It specifies the operation you want. Select from the following values. ?FULA -- Unlocks all locks that the process holds on this channel. If you don't select ?FULA, AOS/VS unlocks only the lock specified by offset ?FLSEL. ?FWFL -- Unlocks the whole file.
?FLCHN	Supply the channel number for the file.
?FLRSW	Reserved. (Set to 0.)
?FLSEL (double-word)	Object number. Your process and cooperating processes agree on the meaning of this number.
?FLPID (double-word)	Reserved. (Set to 0.) ?FLOCK uses this offset.

* There is no default unless otherwise specified.

?GACL *[packet address]*
error return
normal return

Operating System Differences

AOS/RT32 always returns "+,OWARE" as the ACL.

Input

AC0	One of the following: • Byte pointer to the pathname of the target file • 0 if a packet address is supplied
AC1	Byte pointer to a receive buffer for the ACL
AC2	One of the following: • Reserved (Set to 0 if not supplying a packet.) • Address of the ?GACL packet, unless you specify the address as an argument to ?GACL

Output

AC0	Unchanged
AC1	Unchanged
AC2	Unchanged

Error Codes in AC0

ERRAD Read access denied
ERVBP Invalid byte pointer passed as a system call argument
ER_FS_DIRECTORY_NOT_AVAILABLE
Directory not available because the LDU was force released (AOS/VS II only)

Why Use It?

Because ?GACL returns a file or directory's current ACL, you can use it along with ?SACL, which sets the current ACL, or with ?DACL, which sets the default ACL.

Who Can Use It?

There are no special process privileges needed to issue this call. If you specified the file with a channel number, your process must have Read or Write access to the target entry's parent directory or Owner access to the target entry. If you specified the file with a pathname, you must also have Execute access to the target entry's parent directory.

What It Does

?GACL returns the access control list (ACL) associated with a file or directory. AOS/RT32 always returns "+,OWARE" as the ACL.

Before you issue ?GACL, set up a receive buffer for the ACL. The symbol ?MXACL represents the maximum byte length of the buffer. The target file or directory can be specified in one of two ways: either by a byte pointer to the target entry, or by using offset ?GCPCN in the ?GACL packet. The use of a packet is only necessary if you choose to specify the target entry's channel number. AC0 must contain zero to indicate the use of a packet. Load AC1 with a byte pointer to the receive buffer before you issue ?GACL.

The operating system returns the ACL to the receive buffer in the format used for ?DACL and ?SACL. This format is

username<0>accesstype[...]<0>

For example, if the CLI command

ACL FOO BRIAN,OAR MIKE,RE

has executed, then issuing ?GACL to file FOO results in the receive buffer's containing the following 14 bytes in its leftmost bytes.

BRIAN<0><?FACO+?FACA+?FACR>MIKE<0><?FACR+?FACE><0>

Each of symbols <?FACO+?FACA+?FACR> and <?FACR+?FACE> requires 1 byte.

Figure 2–54 shows the structure of the ?GACL packet.

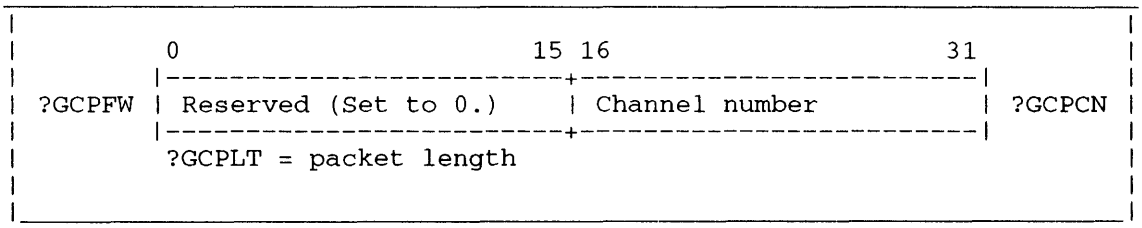


Figure 2–54. Structure of ?GACL Packet

Notes

- See the descriptions of ?DACL and ?SACL in this chapter.

?GBIAS — Gets the current bias factor values.

?GBIAS
error return
normal return

Input

None

Output

AC0 Contains the following:

- Bits 16 through 23 contain the maximum bias factor
- Bits 24 through 31 contain the minimum bias factor

AC1 Undefined

AC2 Undefined

Error Codes in AC0

No error codes are currently defined.

Why Use It?

?GBIAS lets you determine your system's maximum and minimum bias factors at runtime. You can use ?GBIAS in conjunction with ?SBIAS, which sets the bias factor values.

Who Can Use It?

There are no special process privileges needed to issue this call, and there are no restrictions concerning file access.

What It Does

?GBIAS returns the maximum and minimum bias factors for your system to the low–order bits of AC0. If there is no maximum bias factor, the operating system returns 0 to AC0. The default minimum bias factor is zero. ?SBIAS can change it.

Notes

- See the description of ?SBIAS in this chapter.

?GCHR
error return
normal return

Input

AC0	One of the following: • Byte pointer to the name of the target device • Channel (number) on which the device is open	
AC1	Flag bits: Bit 0 = 0 if AC0 contains a byte pointer Bit 0 = 1 if AC0 contains a channel number Bit 1 = 0 to get the current characteristics of the device Bit 1 = 1 to get the default characteristics of the device	
AC2	Address of a 3–word block to receive the device characteristics packet. (See Table 2–29.)	

Output

AC0	Unchanged
AC1	Unchanged
AC2	Unchanged

Error Codes in AC0

ERICN	Illegal channel number
ERIFT	Illegal file type
ERVBP	Invalid byte pointer passed as a system call argument
ERVWP	Invalid word pointer passed as a system call argument

Why Use It?

?GCHR can be a useful preliminary call for ?SCHR, which sets a character device's characteristics. You can also set what are known as "extended characteristics" with ?SECHR and use ?GECHR to obtain them. The ?SECHR and ?GECHR system calls are functional supersets of ?GCHR and ?SCHR. ?GECHR can access all the settings included in the ?GCHR system call. For ease of use, words zero through two, documented in Table 2–29 (page 2-178), have also been consolidated into Table 2–32 (starting on page 2-194) in the ?GECHR system call. Table 2–32 contains a complete listing of the characateristics words.

Who Can Use It?

This call may be issued with and without special process privileges, resulting in different access to files. Some features of the call differ between AOS/VS and AOS/VS II and require special process privileges. These features are described below.

?GCHR Continued

What It Does

The ?GCHR call returns the current or default characteristics assigned to a named character device. Clear bit 1 of AC1 to obtain the named device's current characteristics; set bit 1 of AC1 to obtain the named device's default characteristics assigned by the operating system. Since you also indicate the length of the characteristics word packet in AC1, you can use this call to obtain any number of characteristic words up to ?CLMAX. The system will return the number of characteristic words specified by AC1 to the buffer specified by AC2. Refer to PARU.32.SR for the characteristic's assembled values.

You can use the ?GCHR call to obtain your own current or default device characteristics without any special process privileges. But, to obtain another user's default device characteristics, you must also be PID 2 or a user with the System Manager privilege turned on.

You can obtain any user's current *or* default device characteristics in AOS/VS II, provided you are PID 2 or a user with the System Manager privilege turned on. In AOS/VS you cannot obtain another user's current device characteristics. The operating system returns the characteristics of the target device to the 3–word buffer you specify in AC2. (See Table 2–29.)

Table 2–29. Character Device Characteristics Words

Word	Characteristic	Meaning
0	?CST	Simulates 8-column tabs.
	?CSFF	Simulates form feeds.
	?CEPI	Requires even parity on input.
	?C8BT	Allows 8 data bits per character. You must set this characteristic when you need all 8 bits of each byte when you perform a binary read.
	?CSPO	Sets even parity on output.
	?CRAF	Sends 17. rubout characters after each form feed, unless the form feed is simulated.
	?CRAT	Sends two rubout characters after each tab, unless the tabs are simulated.
	?CRAC	Sends two rubout characters after each carriage return or New Line.
	?CNAS	Non-ANSI-standard device.
	?COTT	The device converts input values of octal 175 and octal 176 to octal 33 (ESC) when it receives them. The device converts octal 175 and octal 176 to octal 33 (ESC) when it sends them.
	?CEOL	No automatic carriage return or line feed at end of each line. (If ?CWRP is off and ?CEOL is on, then the output truncates at the line length defined by the device characteristics.)

(continued)

Table 2–29. Character Device Characteristics Words

Word	Characteristic	Meaning
0 (cont.)	?CUCO	Uppercase output only.
	?CMRI	Monitors ring indicator on modem-controlled line.
	?CFF	Form feed on open.
	?CEB0, ?CEB1	Echo modes (one of the following). No entry = 0B(?CEB0)!0B(?CEB1) results in no echoing. ?CEOS = 0B(?CEB0)!1B(?CEB1) results in echoing exactly as input. (CLI CHARACTERISTIC switch /EB1) ?CEOC = 1B(?CEB0)!0B(?CEB1) results in echoing control characters with uparrows (e.g., ^A) and in echoing ESC as $. (CLI CHARACTERISTIC switch /EB0) 1B(?CEB0)!1B(?CEB1) is reserved; don't use it.
1	?CULC	Both lowercase and uppercase terminal.
	?CPM	Page mode.
	?CNRM	Disables message receipt (accept no ?SEND system calls). The PID 2 process or a process with the System Manager privilege turned ON can override ?CNRM.
	?CMOD	Modem control.
	?CDT0+?CDT3	Device type. See Table 2-30.
	?CTO	Enables device time-outs.
	?CTSP	Retains trailing blanks (card readers only).
	?CPBN	Right-justifies each column to 16-bits (card readers only).
	?CESC	Each ESC character generates a Ctrl-C Ctrl-A interrupt.
	?CWRP	(Hardware) Wraparound for lines longer than the characters per line defined in the last characteristic word.

(continued)

Table 2–29. Character Device Characteristics Words

Word	Characteristic	Meaning
1 (cont.)	?CFKT	Uses function keys as input delimiters (device types ?CRT3 and ?CRT6 only). The two characters that a function key generates go at the end of the data that is in the read buffer. For example, suppose the read buffer is 30 bytes long and the user has typed 14 characters. The user also types Ctrl-H to move the cursor to the left on his screen. Typing a function key then places two characters in bytes 15 and 16 and terminates the read.
	?CNNL	Does not append New Line characters (card readers only).
		Bit 15 is used by TRA/TPA in PARU.16.SR
2	Left byte	Number of lines per page (for Page mode devices).
	Right byte	Characters per line (CPL).

(concluded)

Additional Character Device Characteristics Words

For information about characteristics words 3 through ?CLMAX, look at file PARU.32.SR. This information begins after the comment statement

```
;       ?CH4 - offset 3
```

Several values in offsets 3 through ?CLMAX control the way the operating system supports modems. For example, some modem controllers let you enable their Clear To Send option by your supplying value ?HRDFLC in offset ?CH4 or by your setting the /HOFC switch in a CLI CHARACTERISTICS command. For an explanation of modem support and corresponding values to specify modem support, see the manual *Managing AOS/VS and AOS/VS II*.

Common Character Device Characteristics Words

Table 2–30 explains several of the more common device characteristics found in Table 2–29.

Table 2–30. Commonly Used Device Characteristics

Characteristic	Meaning If Chosen
?CNAS	The device is a non-ANSI-standard terminal. (See "ANSI-Standard Versus Non-ANSI-Standard Terminals" for descriptions of the two types.)
?CULC	Retains uppercase and lowercase characters, rather than converting lowercase characters to their uppercase equivalents.
?CPM	Device is in Page mode. The OS stops the output automatically when the output reaches the number of lines per page you specified or a form-feed character. Form feed frees the output until you release it with Ctrl-Q. (Omit ?CPM if you want form-feed characters to pass normally.)
?CMOD	Modem control. ?CMOD must be set during the system-generation procedure.
?CNNL	Ignores New Line characters. ?CNNL applies to card readers only.
?CDT0 ?CDT1 ?CDT2 ?CDT3	Device type. The OS defines a device's type based on the device's handling of the ESC character and certain control characters. The following bit masks define the device types: ?TTY -- 4010A hard-copy terminal. ?CRT1 -- 4010I display terminal. ?CRT2 -- 6012 display terminal. ?CRT3 -- DASHER display terminal; 605x display terminal. ?CRT6 -- D400/D450 display terminal; 6130 display terminal. ?CRT7 -- ?CRT15 -- user-defined devices.
?C8BT	For 8-bit ASCII terminals only. ?C8BT passes each byte between an Asynchronous Line Multiplexor (ALM) and the user buffer without modification. This characteristic is illegal for DASHER terminals and for other terminals that use the high-order bit (of the byte) for parity.
?CTSP	For card readers only. ?CTSP retains trailing blanks.
?CPBN	For card readers only. ?CPBN right-justifies each card column into a 16-bit memory word.
?CMRI	Monitor Ring Indicator. ?CMRI must be set during the system-generation procedure.

ANSI–Standard Versus Non–ANSI–Standard Terminals

Bit ?CNAS in the first device characteristics word defines your terminal as ANSI standard or non–ANSI standard. The difference between ANSI–standard and non–ANSI–standard terminals is in how the operating system treats the carriage return (CR), line feed (LF), and New Line keys.

When you type a New Line character on an ANSI–standard terminal, the operating system echoes the New Line character and the terminal performs a line feed and a carriage return.

On non–ANSI–standard terminals, the operating system translates each carriage return to a New Line and each line feed to a carriage return. When the operating system outputs a line feed to a non–ANSI–standard terminal, it precedes the line feed with a carriage return.

Notes

- See the descriptions of ?SCHR, ?SECHR, and ?GECHR in this chapter.
- Refer to PARU.32.SR for the assembled values of the character device characteristics.

?GCLOSE — Closes a file previously opened for block I/O.

?GCLOSE
error return
normal return

Input

AC0	Reserved (Set to 0.)
AC1	Contains the following: • Bit 0 is a flag bit: Bit 0 = 1 to modify status on output Bit 0 = 0 otherwise • Bits 1 through 31 contain the channel number associated with the target file
AC2	Reserved (Set to 0.)

Output

AC0	Undefined
AC1	Contains the following: • Bit 0, if set to 1 on input, contains one of the following on output: 0 if the file was not modified 1 if the file was modified • Bits 1 through 31 are unchanged
AC2	Undefined

Error Codes in AC0

ERACU	Attempt to close unopen channel/device
ERCIU	Channel in use
ERSIM	Simultaneous requests on same channel

Why Use It?

You must use ?GCLOSE, rather than the standard ?CLOSE, to close a file that was previously opened for block I/O. Like ?CLOSE, ?GCLOSE breaks the target file's channel assignment, which allows the operating system to reassign that channel number to another file.

Who Can Use It?

There are no special process privileges needed to issue this call. You must have obtained a channel number, via ?OPEN or ?GOPEN, before issuing ?GCLOSE. Furthermore, only the process that opened a channel to a file can close the file.

What It Does

?GCLOSE closes a file previously opened with ?GOPEN for block I/O operations. If you set bit 0 of AC1 on input, the operating system returns the file's status (modified or not modified) to AC1 on output.

?GCLOSE Continued

If you are performing block I/O on a magnetic tape, and you issue ?GCLOSE, the operating system follows the last block written on the tape file with a logical end–of–tape mark (two consecutive tape marks).

Don't use ?GLOSE to close a file you've opened with ?OPEN. Use ?CLOSE instead.

Notes

- See the description of ?GOPEN in this chapter.

AOS/VS

?GCPN
error return
normal return

Input

AC0	One of the following: • PID of the target process • Byte pointer to the name of the target process • –1 to get the calling process's terminal port number
AC1	One of the following: • 0 if AC0 contains a PID • –1 if AC0 contains a byte pointer Otherwise, the OS ignores AC1
AC2	Reserved (Set to 0.)

Output

AC0	Unchanged
AC1	Global port number of the target process's terminal
AC2	Undefined

Error Codes in AC0

ERFDE	File does not exist
ERPOR	PID is out of range for this process
ERPRH	Attempt to access process not in hierarchy
ERVBP	Invalid byte pointer passed as a system call argument

Why Use It?

?GCPN gets the global port number of a terminal for you, even if you know only the name or PID of its associated process.

Who Can Use It?

There are no special process privileges needed to issue this call, and there are no restrictions concerning file access.

What It Does

?GCPN returns the global port number of the target process's terminal.

Before you issue ?GCPN, load AC0 with the PID of the target process, a byte pointer to the name of the target process, or –1. If you specify –1, the operating system returns the port number of the terminal that is associated with the calling process. If the operating system cannot find the PID or the name that you specified, or if there is no terminal associated with the target process, then it returns either error code ERFDE or error code ERPRH to AC0.

?GCRB — Gets the base of the current resource (16–bit processes only).

?GCRB
normal return

Input

None

Output

AC0	Base address of the current resource
AC1	Undefined
AC2	Undefined

Error Codes in AC0

No error codes are currently defined. If an error occurs, the operating system exits via ?BOMB. The state of AC0 is undefined.

Why Use It?

?GCRB is useful for finding out the logical address of an element in any overlay bound for a multiple overlay area. (?GCRB assumes that you know the element's displacement relative to the overlay area base.)

Who Can Use It?

There are no special process privileges needed to issue this call, and there are no restrictions concerning file access.

What It Does

?GCRB returns a label to AC0 that indicates the base address of the current resource. You can use this information with the resource system calls to determine the current base–relative offset of a movable resource. (A movable resource is an overlay that the operating system can relocate to any basic area within a multiple overlay area.) To determine the absolute address of an offset in a particular resource, add the relative value of that offset to the value of the resource base.

Note that ?GCRB has no error return. If the operating system encounters an invalid resource base, it transfers control to the ?BOMB error–detection routine.

?GDAY
error return
normal return

Input

None

Output

AC0	Day from 1 through 31
AC1	Month from 1 through 12
AC2	Year minus 1900

Error Codes in AC0

No error codes are currently defined.

Why Use It?

You can use ?GDAY to get the current date. Next, you can write it on output page headers.

Who Can Use It?

There are no special process privileges needed to issue this call, and there are no restrictions concerning file access.

What It Does

?GDAY gets the current date from the system clock, and returns it as day, month, and year to AC0, AC1, and AC2, respectively. Notice that the value for the current year is an offset from a base of 1900.

?GDLM
error return
normal return

Input

AC0	One of the following: • byte pointer to the device name • channel number of the file
AC1	Flag bits: • ?SDCN if AC0 contains a channel number • ?SDDN if AC0 contains a byte pointer to a device name • ?SDTO to get the output delimiter table • ?SDTI to get the input delimiter table • ?SDTP to get the priority input delimiter table
AC2	Word address of the 16–word area to receive the delimiter table.

Output

AC0	Unchanged
AC1	Unchanged
AC2	Unchanged

Error Codes in AC0

ERFDE	File does not exist
ERICN	Illegal channel number
ERIDP	Illegal destination port
ERIFT	Illegal file type
ERMPR	System call parameter address error (the receive buffer is not in the unshared area of your address space)
ERPRV	Caller not privileged for this action
ERVBP	Invalid byte pointer
ERVWP	Invalid word pointer

Why Use It?

?GDLM is useful whenever you want to save the current delimiter table. For example, you may need to set up a temporary delimiter table for a special situation, but later revert to the original delimiter table.

Who Can Use It?

There are no special process privileges needed to issue this call, and there are no restrictions concerning file access.

What It Does

?GDLM returns a delimiter table for a file with data–sensitive records or for a character device. You can set a delimiter table either with ?SDLM or ?OPEN.

Before you issue ?GDLM, reserve a table of 16 consecutive words in your address space and load the word address of the table into AC2. (For information on the structure of delimiter tables, see the description of the ?OPEN system call.)

Notes

- See the descriptions of ?SDLM and ?OPEN in this chapter.

?GECHR
error return
normal return

Input

AC0 contains one of the following:

- Byte pointer to the name of the target device
- Channel (number) on which the device is open

AC1 contains the following flags:

Bit 0 = 0	If AC0 contains a byte pointer
Bit 0 = 1	If AC0 contains a channel number
Bit 1 = 0	To get the current characteristics
Bit 1 = 1	To get the default characteristics
Bits 28 – 31	Length of the packet in words, between values ?CLMIN and ?CLMAX inclusive

AC2 contains the address of a ?CLMIN to ?CLMAX word range to receive the device characteristics packet.

Output

AC0 Unchanged

AC1 Number of characteristic words returned

AC2 Unchanged

Error Codes in AC0

ERICN	Illegal channel
ERIFT	Illegal file type
ERPRE	Invalid system call parameter
ERARG	Too few or too many arguments to PMGR. You attempted to define a packet with length less than ?CLMIN.
ERVBP	Invalid byte pointer passed as a system call argument
ERVWP	Invalid word pointer passed as a system call argument

Why Use It?

You can use the ?GECHR call to examine the current characteristics before using ?SECHR to set a character device's characteristics. Suppose an attempt is made to get characteristics on a device where the characteristics have no meaning, (e.g., the fourth characteristic word is only valid on a system with intelligent controllers that can deal with Baud rates in real time — ECLIPSE MV/8000®–II, ECLIPSE MV/6000®, ECLIPSE MV/10000®, ECLIPSE MV/2000®). Then, a bit (?CVAL) will be set to 0 in the fourth word to indicate the word is invalid in the current configuration. The ?GECHR and ?SECHR system calls are functional supersets of ?GCHR and ?SCHR respectively. The ?GECHR call can obtain all the settings obtained by the ?GCHR call. You can use the ?GECHR call as a replacement for the ?GCHR call, when the packet length specified in AC1 is 3.

For completeness and ease of use, words zero through two, documented for the ?GCHR call in Table 2–29 (starting on page 2-178), have been consolidated into Table 2–32.

Who Can Use It?

This call may be issued with and without special process privileges resulting in different access to files. Some features of the call differ between AOS/VS and AOS/VS II and require special process privileges. These features are described below.

What It Does

The ?GECHR call returns the current or default characteristics assigned by the operating system to the named character device. Clear bit 1 of AC1 to obtain the named device's current characteristics; set bit 1 to obtain the named device's default characteristics. Since you also indicate the length of the characteristics word packet in AC1, you can use this call to obtain any number of characteristic words up to ?CLMAX. The operating system will return the number of characteristic words specified by AC1 to the buffer specified by AC2. Refer to PARU.32.SR for the characteristic's assembled values.

You can use the ?GECHR call to obtain your own current or default device characteristics without any special process privileges. But, to obtain another user's default device characteristics, you must also be PID 2 or a user with the System Manager privilege turned on.

You can obtain any user's current *or* default device characteristics in AOS/VS II, provided you are PID 2 or a user with the System Manager privilege turned on. In AOS/VS you cannot obtain another user's current device characteristics.

The device characteristics packet parameter offset values for characteristics one (offset 0) through thirteen (offset 12), and their characteristic packet size symbols, are listed in Table 2–31. Character device characteristics words zero through two and their symbols in the characteristics packet are defined in the ?GCHR description in Table 2–29 (page 2-178). Character device characteristics words zero through thirteen and their symbols, are fully defined in Table 2–32.

The rubout, echo, cursor control, and control key characteristics for different device types are listed in Table 2–33.

The ?BMDEV characteristic indicates that the targeted device is a bit–mapped device, which implies that you can create windows on this device.

The ?CWIN characteristic indicates that the targeted device is actually a window on a physical bit–mapped device. You can use ?CWIN to determine whether you can issue ?GRAPHICS, ?PTRDEVICE, and ?WINDOW system calls.

The ?TRPE characteristic indicates that pointer events (e.g., mouse movements) delimit/terminate ?READ system calls. If you are interested in retrieving pointer events, then set this characteristic via ?SECHR.

Setting the ?CXLT characteristic enables VT100, VT220, and DG ANSI terminal modes. DG ANSI terminal mode is supported in all environments except MCP1, CPI–24, IAC–8 (for non 68K based processors), IAC16, LAC–12 and MV/7500DC duarts.

The default setting for the ?CNLX characteristic is zero with Kanji language translation disabled. Setting the ?CNLX characteristic activates host–based Kanji language translation for a particular console. On PC terminals which run their own language translation software, setting the ?CNLX bit while the user is entering translated text can result in unpredictable results.

In AOS/VS II only, setting the ?CKVT and the ?CXLT characteristics enables support for Kanji VT100 compatible terminals. You can also turn on the ?CKVT characteristic with the CLI. The ?C16B characteristic enables double–byte character handling as a single unit for the Kanji character sets.

In AOS/VS II only, for all Kanji character sets except the Taiwanese, double byte characters are formed using two 8–bit bytes, each within the range 241–376 octal. This range is known as the G1 character set. The range from 41–176 octal is known as the G0 character set.

Some Taiwanese characters' second 8–bit bytes are drawn from both the G0 and the G1 character sets. Provided ?C16B has also been set, the ?CKGO characteristic enables recognition of double byte characters in the the G1–G0 character set. The ?CKGO characteristic is normally off for Japanese Kanji applications, and on for Taiwanese ones.

Intelligent asynchronous controllers allocate a portion of their memory, sometimes called the ring buffer, for input and output buffering. The amount of controller memory varies from revision to revision, and is accounted for in the default value settings in VSGEN. But if you have used larger than default values in VSGEN, or if you have selected Asian Language or DG ANSI mode support, on some controllers your system may try to use more controller memory space than actually exists. Your system will come up, but the controller will not function properly. Refer to the "Notes and Warnings" section of your release notice for exact information about available memory aboard such controllers.

Some controllers, modems, and printers use hardware flow control. Use the ?HRDFLC characteristic to enable or disable hardware flow control, or you can use the CLI CHARACTERISTICS command. Consult the VSGEN chapter in *Installing, Starting, and Stopping AOS/VS II* or in *Installing, Starting, and Stopping AOS/VS* for more information about hardware flow control.

Notes

- See the descriptions of ?GCHR, ?SCHR, and ?SECHR in this chapter.

Table 2–31. Characteristics Packet Parameters

Word	Characteristic	Meaning
1	?CH1 = 0.	Offset 0.
2	?CH2 = 1.	Offset 1.
3	?CH3 = 2.	Offset 2.
4	?CH4 = 3.	Offset 3.
5	?CH5 = 4.	Offset 4.
6	?CH6 = 5.	Offset 5.
7	?CH7 = 6.	Offset 6.
8	?CH8 = 7.	Offset 7.
9	?CH9 = 8.	Offset 8.
10	?CH10 = 9.	Offset 9.
11	?CH11 = 10.	Offset 10.
12	?CH12 = 11.	Offset 11.
13	?CH13 = 12.	Offset 12.
14	?CH14 = 13.	Offset 13.
		Packet length parameters:
	?CLMIN = 3.	Minimum length of the packet.
	?CLMAX = 15.	Maximum length of the packet.

Table 2–32. Character Device Characteristics Words and Symbols

Word	Characteristic	Meaning
0	?CST	Simulates 8-column tabs.
	?CSFF	Simulates form feeds.
	?CEPI	Requires even parity on input.
	?C8BT	Allows 8 data bits per character. You must set this characteristic when you need all 8 bits of each byte when you perform a binary read.
	?CSPO	Sets even parity on output.
	?CRAF	Sends 17. rubout characters after each form feed, unless the form feed is simulated.
	?CRAT	Sends two rubout characters after each tab, unless the tabs are simulated.
	?CRAC	Sends two rubout characters after each carriage return or New Line.
	?CNAS	Non-ANSI-standard device.
	?COTT	The device converts input values of octal 175 and octal 176 to octal 33 (ESC) when it receives them. The device converts octal 175 and octal 176 to octal 33 (ESC) when it sends them.
	?CEOL	No automatic carriage return or line feed at end of each line. (If ?CWRP is off and ?CEOL is on, then the output truncates at the line length defined by the device characteristics.)
	?CUCO	Uppercase output only.
	?CMRI	Monitors ring indicator on modem-controlled line.
	?CFF	Form feed on open.
	?CEB0, ?CEB1	Echo modes (one of the following). No entry = 0B(?CEB0)!0B(?CEB1) results in no echoing. ?CEOS = 0B(?CEB0)!1B(?CEB1) results in echoing exactly as input. (CLI CHARACTERISTIC switch /EB1)

(continued)

Table 2–32. Character Device Characteristics Words and Symbols

Word	Characteristic	Meaning
0 (cont.)		Echo modes (continued). ?CEOC = 1B(?CEB0)!0B(?CEB1) results in echoing control characters with uparrows (e.g., ^A) and in echoing ESC as $. (CLI CHARACTERISTIC switch /EB0) 1B(?CEB0)!1B(?CEB1) is reserved; don't use it.
1	?CULC	Both lowercase and uppercase terminal.
	?CPM	Page mode.
	?CNRM	Disables message receipt (accept no ?SEND system calls). The PID 2 process or a process with the System Manager privilege turned ON can override ?CNRM.
	?CMOD	Modem control.
	?CDT0+?CDT3	Device type (see Table 2-30, page 2-181.).
	?CTO	Enables device time-outs.
	?CTSP	Retains trailing blanks (card readers only).
	?CPBN	Right-justifies each column to 16-bits (card readers only).
	?CESC	Each ESC character generates a Ctrl-C Ctrl-A interrupt.
	?CWRP	(Hardware) Wraparound for lines longer than the characters per line defined in the last characteristic word.
	?CFKT	Uses function keys as input delimiters (device types ?CRT3 and ?CRT6 only). The two characters that a function key generates go at the end of the data that is in the read buffer. For example, suppose the read buffer is 30 bytes long and the user has typed 14 characters. The user also types Ctrl-H to move the cursor to the left on his screen. Typing a function key then places two characters in bytes 15 and 16 and terminates the read.
	?CNNL	Does not append New Line characters (card readers only). Bit 15 is used by TRA/TPA in PARU.16.SR

(continued)

Table 2–32. Character Device Characteristics Words and Symbols

Word	Characteristic	Meaning
2	Left byte	Number of lines per page (for Page mode devices).
	Right byte	Characters per line (CPL).
3	?CVAL = 0.	Indicates that the contents of this offset are valid (used on return from ?GECHR.) In general, ?CVAL= 1 for an IAC system, and ?CVAL = 0 otherwise.
	?BR0BIT	First baud rate field (Bit 0, of the field)
	?CTCK	Internal transmitter clock; used by VSGEN for controlling split baud.
	?CRCK	Internal receiver clock; used by VSGEN for controlling split baud.
	?BR1BIT	Second baud rate field (Bit 1).
	?BR2BIT	Third baud rate field (Bit 2).
	?BR3BIT	Fourth baud rate field (Bit 3).
	?BR4BIT	Fifth baud rate field (Bit 4).
		Define baud rate in the 5-bit field with bit mask ?BRMSK.
		The mask offset symbols and their corresponding baud rates are:
		Symbol — Baud
		?CR50 = 0B(?BR0BIT)+0.B(?BR4BIT) 50
		?CR75 = 0B(?BR0BIT)+1.B(?BR4BIT) 75
		?CR110 = 0B(?BR0BIT)+2.B(?BR4BIT) 110
		?CR134 = 0B(?BR0BIT)+3.B(?BR4BIT) 134.5
		?CR150 = 0B(?BR0BIT)+4.B(?BR4BIT) 150
		?CR300 = 0B(?BR0BIT)+5.B(?BR4BIT) 300
		?CR600 = 0B(?BR0BIT)+6.B(?BR4BIT) 600
		?CR12H = 0B(?BR0BIT)+7.B(?BR4BIT) 1200
		?CR18H = 0B(?BR0BIT)+8.B(?BR4BIT) 1800
		?CR20H = 0B(?BR0BIT)+9.B(?BR4BIT) 2000
		?CR24H = 0B(?BR0BIT)+10.B(?BR4BIT) 2400
		?CR36H = 0B(?BR0BIT)+11.B(?BR4BIT) 3600
		?CR48H = 0B(?BR0BIT)+12.B(?BR4BIT) 4800
		?CR72H = 0B(?BR0BIT)+13.B(?BR4BIT) 7200
		?CR96H = 0B(?BR0BIT)+14.B(?BR4BIT) 9600
		?CR19K = 0B(?BR0BIT)+15.B(?BR4BIT) 19200
		?CR45 = 1B(?BR0BIT)+0.B(?BR4BIT) 45.5
		?CR38K = 1B(?BR0BIT)+1.B(?BR4BIT) 38400
		Values 2 to 15, are reserved. (Set to 0.)

(continued)

Table 2–32. Character Device Characteristics Words and Symbols

Word	Characteristic	Meaning
3 (cont.)	?CST0+?CST1	Define these stop bits with the stop bit mask ?CSMSK. Other symbols are: ?CS10 = 0B?CST0+1B?CST1 1 stop bit. ?CS15 = 1B?CST0+0B?CST1 1.5 stop bits. ?CS20 = 1B?CST0+1B?CST1 2 stop bits.
	?CPTY ?CPEN	Odd/Even parity. Parity Disabled/Enabled. Set the parity bit field values with the parity field mask ?CPMSK. The mask symbol's corresponding parity settings are: ?CPR0 = 0B?CPEN disables parity checking ?CPR1 = 1B?CPEN+0B?CPTY enables odd parity ?CPR2 = 1B?CPEN+1B?CPTY enables even parity.
	?CLT0+?CLT1	Data field length bits. Use the mask ?CLMSK to set the data field length in bits. For example, ?CLMSK = 1B?CLT0+1B?CLT1 is an 8-bit data field length. Data field lengths are: ?CLN5 = 0B?CLT0+0B?CLT1 5 data bits. ?CLN6 = 0B?CLT0+1B?CLT1 6 data bits. ?CLN7 = 1B?CLT0+0B?CLT1 7 data bits. ?CLN8 = 1B?CLT0+1B?CLT1 8 data bits.
	?BRFCT	Baud rate factor, 16x; used by VSGEN for split baud.
	?HRDFLC	Hardware Flow Control (CTS). ?CHOFC = ?HRDFLC is hardware output flow control.

(continued)

Table 2–32. Character Device Characteristics Words and Symbols

Word	Characteristic	Meaning
4	?SHCO	Shared console ownership characteristic.
	?XOFC	XON XOFF output flow control.
	?XIFC	XON XOFF input flow control.
	?C16B	Enable double byte handling (16 bit characters).
	?BMDEV	Bitmap device.
	?TRPE	Terminate read on a pointer event.
	?CWIN	Window characteristic.
	?CACC	Enforce access control.
	?CCTD	Port is in a contended environment (PBX, Termserver).
	?CSRDS	Suppress Receiver Disable.
	?CXLT	Translate VT100, VT220, and ANSI terminals.
	?CABD	Autobaud matching if set to 1.
	?CALLOUT	Call out, for PBX support.
	?CBK0	Break function bit 0.
	?CBK1	Break function bit 1.
	?CBK2	Break function bit 2.
		Use the mask ?CBKM to set the break field bits. For example, ?CBKM = 1B?CBK0+1B?CBK1+1B?CBK2 selects all the break bits. The break field definitions are:
		?CBBM = 0B(?CBK2) Break binary mode. ?CBDS = 1B(?CBK2) Force disconnect. ?CBCA = 2B(?CBK2) Send ^C^A sequence. ?CBCB = 3b(?CBK2) Send ^C^B sequence. ?CBCF = 4B(?CBK2) Send ^C^F sequence. 5B(?CBK2) Reserved.(Set to 0.) 6B(?CBK2) Reserved.(Set to 0.) 7B(?CBK2) Reserved.(Set to 0.)
5	?CMDOP = ?CH6	Modem options.
		?CDMC = 0 Reserved (Set to 0.) ?CMDUA = 1 Direct user access to modem (don't pend the first write). ?CHDPX = 2 Half duplex. ?CSMCD = 3 Suppress monitoring CD (Carrier Detect) for a modem connection. ?CRTSCD = 4 On half duplex, don't raise RTS until CD drops. ?CHIFC = 5 Hardware input flow control.

(continued)

Table 2–32. Character Device Characteristics Words and Symbols

Word	Characteristic	Meaning
6	?CTCC = ?CH7	Msec to wait for CD on a modem connection.
7	?CTCD = ?CH8	Msec to wait for CD, if it drops.
8	?CTDW = ?CH9	Msec to wait after connection before allowing I/O.
9	?CTHC = ?CH10	Msec to wait after a disconnect so the modem will settle.
10	?CTLT = ?CH11	Msec to wait before turning the line around (from transmit to receive) for half duplex.
11	?CCTYPE = ?CH12	Console type. The high byte is reserved. (Set to 0.) The low byte is console type. Use the mask ?CCTYPMSK for accessing only console type. For example, ?CCTYPMSK = 377 is a mask for console type. The current values for console types are: ?CDCC = 0 direct connection. ?CLNC = ?CDCC+1 TermServer connection. ?CTNC = ?CLNC+1 TELNET consoles. ?CPDC = ?CTNC+1 PAD consoles. ?CVRC = ?CPDC+1 virtual (SVTA-like) consoles. ?CPXC = ?CVRC+1 PBX consoles (PIM). ?CPCC = ?CPXC+1 PC/TS consoles. ?CBMC = ?CPCC+1 bitmapped (Windowing) console ?CTPC = ?CBMC+1 T1 Primary Rate console (IWC)
12	?CLFP = ?CH13	Language Front-end Processor (LFP). The LFP options are: ?CKG0 = 0 G1-G0 double-byte handling. ?CKHW = 1 Kanji half-wide characters. ?CNLX = 2 native language translation. ?CKVT = 3 Kanji VT100 support, AOS/VS II only.

(concluded)

Table 2–33. Device Types for Rubout Echo and Cursor Controls

Device Type	Model Number	Characters' Effect: Move Left:	Move Right:	Erase Line:	Rubout Echo:
0	4010A	(None)	(None)	(None)	SHIFT O
0	6040	(None)	(None)	(None)	SHIFT O
1	4010I	^Z	^Y	^K	^Z,SPACE,^Z
2	6012	^Y	^X	^K	^Y,SPACE,^Y
3	6052	^Y	^X	^K	^Y,SPACE,^Y
4	----	ESC,D	ESC,C	ESC,K	ESC,D,SPACE,ESC,D
5	----	----	----	----	----
6	6130	^Y	^X	^K	^Z,SPACE,^Z
7-15	(For future expansion)				

?GHRZ — Gets the frequency of the system clock.

?GHRZ
error return
normal return

Input

None

Output

AC0 One of the following code values, which indicates the system clock's frequency:

Code	Frequency (Hz)
0	60
1	10
2	100
3	1000
4	50

AC1 Undefined

AC2 Undefined

Error Codes in AC0

No error codes are currently defined.

Why Use It?

Your application may require you to know your system clock's real–time frequency. To get this information, you can use ?GHRZ.

Who Can Use It?

There are no special process privileges needed to issue this call, and there are no restrictions concerning file access.

What It Does

?GHRZ returns the system clock's real–time frequency, which was set during the system–generation procedure. The operating system returns the frequency as one of the code values listed above. (Real–time clock frequencies are measured in hertz (Hz).)

Notes

- Refer to the chapter about VSGEN in the manual *Installing, Starting, and Stopping AOS/VS II* for information on setting the system clock's real–time frequency.

?GLINK
error return
normal return

Input		Output	
AC0	Byte pointer to the link entry's pathname	AC0	Unchanged
AC1	Byte pointer to a receive buffer for the resolved pathname	AC1	Unchanged
AC2	Reserved (Set to 0.)	AC2	Undefined

Error Codes in AC0

ERIFT	Illegal file type (The target entry is not a link entry.)
ERRAD	Read access denied
ERVBP	Invalid byte pointer passed as a system call argument

Why Use It?

You can use ?GLINK to decide whether to delete an existing link entry and/or create a new one.

Who Can Use It?

There are no special process privileges needed to issue this call. However, you must have Read and Execute access to the link entry's parent directory.

What It Does

?GLINK returns the contents of the target link entry to the receive buffer that you specify in AC1.

Before you issue ?GLINK, set up a receive buffer in your logical address space. (The symbol ?MXPL represents the maximum byte length of the buffer. File PARU.32.SR defines the value of this symbol.) Next, load the accumulators with the appropriate values. Then, issue ?GLINK. If the pathname that you specify in AC0 is not the pathname of a link entry, ?GLINK fails and the operating system returns error code ERIFT to AC0.

?GLIST
error return
normal return

Input

AC0	Reserved (Set to 0.)
AC1	Byte pointer to a receive buffer for the search list
AC2	Byte length of the receive buffer

Output

AC0	Undefined
AC1	Unchanged
AC2	Unchanged

Error Codes in AC0

ERVBP Invalid byte pointer passed as a system call argument
ER_FS_DIRECTORY_NOT_AVAILABLE
Directory not available because the LDU was force released (AOS/VS II only)

Why Use It?

?GLIST lets you examine the current contents of your search list. Thus, you can use ?GLIST as a preliminary system call to ?SLIST, which changes the contents of the search list.

Who Can Use It?

There are no special process privileges needed to issue this call, and there are no restrictions concerning file access.

What It Does

?GLIST returns a copy of the calling process's search list to the buffer that you specify in AC1.

Before you issue ?GLIST, load AC1 with a byte pointer to the receive buffer and load AC2 with the length of the buffer. (The symbol ?MXPL represents the maximum length of the search list buffer.) Then, after you issue ?GLIST, the operating system returns the search list to the buffer in the following format:

pathname<0>...pathname<0><0>

where

pathname is a complete pathname string.

<0> is the null terminator.

Be sure to allocate enough buffer space to accommodate the two trailing nulls that the operating system appends to the search list.

Notes

- See the description of ?SLIST in this chapter.

?GMEM Returns the number of undedicated memory pages.

?GMEM
error return
normal return

Input

None

Output

AC0 Undefined

AC1 Current number of undedicated memory pages

AC2 Undefined

Error Codes in AC0

No error codes are currently defined.

Why Use It?

?GMEM gives you information about overall memory contention. In general, the smaller the value returned by ?GMEM, the more likely memory contention becomes. Issuing ?GMEM before you wire pages to a resident process's working set helps you to correctly adjust the size of the resident process to your memory configuration. Do not wire more pages than the current number of undedicated memory pages, because it can cause a system deadlock.

Who Can Use It?

There are no special process privileges needed to issue this call, and there are no restrictions concerning file access.

What It Does

?GMEM returns the current number of undedicated pages available to the calling process. Undedicated pages are unused pages that the operating system can allocate to the calling process.

Notes

- See the description of ?WIRE in this chapter.

?GNAME
error return
normal return

Input

AC0	Byte pointer to a filename or pathname fragment
AC1	Byte pointer to a receive buffer for the pathname
AC2	Byte length of the receive buffer

Output

AC0	Unchanged
AC1	Unchanged
AC2	Byte length of the complete pathname, excluding the null terminator

Error Codes in AC0

ERFAD	File access denied
ERFDE	File does not exist
ERIRB	Insufficient room in buffer
ERMPR	System call parameter address error
ERVBP	Invalid byte pointer passed as a system call argument

Why Use It?

?GNAME can be useful as a preliminary system call for system calls that require a complete pathname as input.

Who Can Use It?

Required file access privileges depend on whether your program runs under the old file system (before AOS/VS II Release 1.00 and with any revision of AOS/RT32 or of AOS/VS) or under the new file system (effective with AOS/VS II Release 1.00).

Old File System

There are no special process privileges needed to issue this call. The calling process must have access (of any type) to each file in the pathname fragment or have Read access to their parent directories. The calling process must also have Execute access to the files' parent directories. However, the operating system does not check for access privileges if the pathname fragment consists of only a prefix (for example, =).

New File System

There are no special process privileges needed to issue this call. The calling process must have access (of any type) to each file in the pathname fragment or have Read access to their parent directories. The calling process must also have Execute access to the files' parent directories. However, the operating system does check for access privileges in all cases, including when the pathname fragment consists of only a prefix (for example, =). In this case, the calling process still must have access (of any type) to each file in the pathname fragment or have Read access to their parent directories.

An Important Difference

One important difference between issuing ?GNAME under the old and new file systems occurs in the following sequence of events:

- The CLI issues ?PROC to create a new child CLI process.
- The newly created child CLI process issues ?SUSER and ?SUPROC to explicitly disable Superuser and Superprocess mode in case the parent process has either privilege.
- The child CLI process issues ?GNAME against "=" to determine the full pathname of the initial directory. Furthermore, suppose the child process does not have access to its working directory, "=".
  - Under the old file system ?GNAME succeeds because AOS/VS performs no access checking; however, the child CLI process has no access to its initial directory. If this process changes its working directory and tries to reenter the initial directory (via, say, a **DIR/** command), ?GNAME returns the error ERFAD.
  - Under the new file system ?GNAME fails right away with error ERFAD. The child CLI process terminates.

What It Does

?GNAME returns the complete pathname, starting with the root, when you specify a pathname fragment in AC0. ?GNAME functions with either an open or a closed file. If you load AC0 with the byte pointer to the prefix =, the operating system returns the pathname of the current working directory.

You cannot use ?GNAME to obtain the "true" pathname associated with a generic file. For example, ?GNAME would return a complete pathname of :PER:DATA for the input pathname @DATA, even though the "true" complete pathname is really :UDD:USER:DATA. Therefore, to obtain the "true" pathname, use ?GRNAME.

Notes

- See the description of ?GRNAME in this chapter.

?GNFN *[packet address]*
error return
normal return

Input

AC0	Reserved (Set to 0.)
AC1	Channel number of the target directory
AC2	Address of the ?GNFN packet, unless you specify the address as an argument to ?GNFN

Output

AC0	Undefined
AC1	Unchanged
AC2	Address of the ?GNFN packet

Error Codes in AC0

ERCIU	Channel in use
EREOF	End of file (There are no more files in the target directory.)
ERFNO	Channel not open
ERFTL	Filename too long (The template exceeds 63 characters.)
ERIFC	Illegal filename character
ERNDR	Not a directory (The channel that you specified is not opened on a directory.)
ERVBP	Invalid byte pointer passed as a system call argument
ERVWP	Invalid word pointer passed as a system call argument
ER_FS_DIRECTORY_NOT_AVAILABLE	Directory not available because the LDU was force released (AOS/VS II only)

Why Use It?

?GNFN allows you to obtain a specific directory's filenames. The CLI, for example, uses ?GNFN to implement its FILESTATUS command.

Who Can Use It?

There are no special process privileges needed to issue this call. However, your process must be able to open (via ?OPEN or ?GOPEN) the directory. It must also have Read and Execute access to the directory at the time it issues ?OPEN or ?GOPEN.

What It Does

?GNFN returns a filename in the target directory (or all filenames that match the template that you specify) to the buffer that you specify in the ?GNFN packet. The first time you issue ?GNFN, the operating system returns the name of the first file in the directory; the second time, it returns the second filename, and so on.

To get all the filenames, use the + template. You can also obtain every filename by issuing repeated ?GNFN system calls until ?GNFN fails on error code EREOF. (EREOF is an end–of–file condition that means you have reached the last file entry in the directory.)

?GNFN Continued

Before you issue ?GNFN, you must open the target directory. After you open the directory with ?OPEN (or ?GOPEN), load AC1 with the correct channel number, which returns in the ?OPEN (or ?GOPEN) packet. You can specify the packet address as an argument to ?GNFN, or you can load it into AC2 before you issue ?GNFN. Figure 2–55 shows the structure of the ?GNFN packet, and Table 2–34 describes its contents.

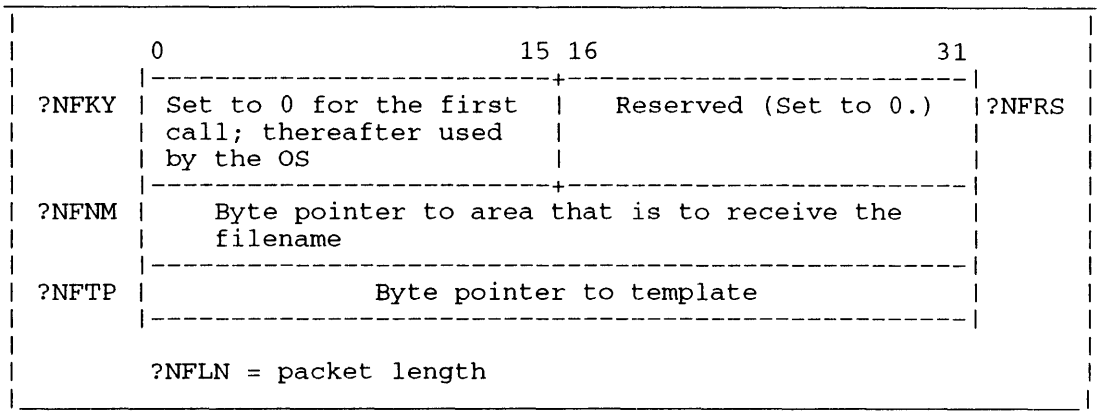


Figure 2–55. Structure of ?GNFN Packet

Table 2–34. Contents of ?GNFN Packet*

Offset	Contents
?NFKY	OS internal value. Set to 0 the first time you issue ?GNFN; ignore thereafter. This value is a key to where the list should continue. So do not set ?NFKY to zero between calls.
?NFRS	Reserved. (Set to 0.)
?NFNM (doubleword)	Byte pointer to area that is to receive the filename.
?NFTP (doubleword)	Byte pointer to filename template, if present. DEFAULT = -1 (no template). If you omit the template, the operating system searches only the open directory. Table 2-35 shows the template characters.

* There is no default unless otherwise specified.

Table 2–35. Filename Template Characters

Character	Description
* (asterisk)	Matches any single filename character except a period.
- (hyphen)	Matches any series of characters not containing a period.
+ (plus sign)	Matches any series of characters.

The first time you issue ?GNFN, set the contents of offset ?NFKY to 0. Do not modify the contents of ?NFKY on subsequent system calls, because the operating system uses this offset as an internal pointer. There is no default value for offset ?NFNM. Therefore, you must use offset ?NFNM as a byte pointer to a receiving area for the filename. If you do not want to use a template, choose the default value (–1) of offset ?NFTP.

Sample Packet

```
PKT:    .BLK    ?NFLN           ;Allocate enough space for the packet
                                ;(packet length is ?NFLN).
        .LOC    PKT+?NFKY       ;Set to 0 for first call. The OS
        .WORD   0               ;uses this value for second and
                                ;subsequent calls.
        .LOC    PKT+?NFNM       ;Byte pointer to an area to receive
                                ;the filename.
        .DWORD  BUFF*2          ;Byte pointer to BUFF.
        .LOC    PKT+?NFTP       ;Byte pointer to template.
        .DWORD  -1              ;No template (default is -1).
        .LOC    PKT+?NFLN       ;End of packet.
```

Notes

- See the description of ?OPEN in this chapter.

?GOPEN *[packet address]*
error return
normal return

Operating System Differences

Only AOS/VS II supports ?ODTL in offset ?ODF1 of the packet extension.

Input

AC0	Byte pointer to the target file's pathname
AC1	Target file's channel number, or −1, to direct the OS to assign a channel number
AC2	Address of the ?GOPEN packet, unless you specify the address as an argument to ?GOPEN

Output

AC0 Contains the following:

- Bits 0 through 25 are reserved (Set to 0.)
- Bits 26 through 31 contain one or more of the following file access privilege masks:

?FACO	Owner access
?FACW	Write access
?FACA	Append access
?FACR	Read access
?FACE	Execute access

AC1 Unchanged

AC2 Address of the ?GOPEN packet

Error Codes in AC0

ERDMO Disk marked as owned by another system
ERDRS Device reserved by another port
ERFTM File/tape mismatch
ERITD Indecipherable tape density
ERVBP Invalid byte pointer passed as a system call argument
ERVWP Invalid word pointer passed as a system call argument
ER_FS_NO_TLA_PRIVILEGE
Attempt to issue ?GOPEN with ?ODTL value supplied, but without proper privilege
ER_FS_OBSOLETE_IPC_FILE_DETECTED
Obsolete IPC file type has been detected; file has been deleted (AOS/VS II only)

Why Use It?

You must use ?GOPEN to open a file for block I/O or physical block I/O. Also, you must issue ?GOPEN to open an MCA as a unit or to open a specific link. Then, if you use the CLI MOUNT command to mount a tape, you must ?GOPEN the file using its link name.

Who Can Use It?

To issue this call, you need a special privilege for only one situation. Your process must be running under the new file system (AOS/VS II) and in Superuser mode in order to issue ?GOPEN with value ?ODTL supplied in offset ?ODF1 of the packet extension.

You must have Read and Execute access to the file's parent directory. In addition, you must have Read access to the file to read from it and Write access to the file to write to it.

What It Does

?GOPEN opens a file for block I/O and assigns the file a channel number. You can specify the channel number in AC1 or force the operating system to assign the channel number by loading AC1 with –1. You can open any of the following devices for block I/O: disk files and units, tape file, MTB tape units, MTD tape units, and MCA units.

The ?GOPEN packets are different for IPC files (see Figure 2–56 and Table 2–36) and for all other file types (see Figure 2–57 and Table 2–38). Although you must reserve sufficient space in your logical address space for the packets, you need to provide input specifications for offset ?OPFL (for the standard packet) or offset ?OPCH (for the IPC packet) only.

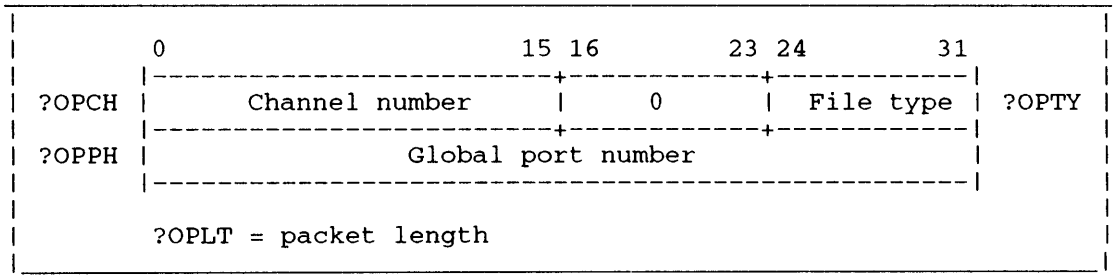


Figure 2–56. Structure of ?GOPEN Packet for IPC Files

Table 2–36. Contents of ?GOPEN Packet for IPC Files*

Offset	Input Value	Output Value
?OPCH	None.	Channel number.
?OPTY	None.	Left byte: 0 Right byte: File entry type (?FIPC)
?OPPH (doubleword)	None.	Global port number.

* There is no default unless otherwise specified.

Table 2–37. Contents of Standard ?GOPEN Packet*

Offset	Input Value	Output Value
?OPFL	Flag bits ?OPME--Exclusive open. ?OPMD--Inhibit initial form feed. ?OPXP--Packet has extension. If tape density mode, specify one of the following: ?OPDL--800 bpi. ?OPDM--1600 bpi. ?OPDH--6250 bpi. ?OPAM--automatic density mode matching selected ?OPD5--Low density ?OPD6--Medium density ?OPD7--High density Transfer mode (for Model 6352 magnetic tape units only): ?OPMBF--Use buffered mode when performing tape I/O ?OPMST--Use streaming mode when performing tape I/O DEFAULT--0 for non-buffered and non-streaming data transfer	Bits 0-15: Channel number.
?OPTY	None.	Left byte: Record format Right byte: File entry type
?OPFC	None.	File control parameters. See the description of offset ?SCPS in the explanation of ?FSTAT.
?OPEW	None. (Set to 0.)	Unchanged.
?OPEH (doubleword)	None	File size in bytes.

* There is no default unless otherwise specified.

The operating system returns all the other parameters, based on the specifications you set when you created the file. Although the operating system returns file type and format information, the block I/O and physical block I/O system calls ignore record formatting information, because they operate in terms of blocks only.

Notice that offset ?OPFL/?OPCH in both packets contains different input and output values. On input, you can use ?OPFL/?OPCH to indicate a tape–density mode if you are opening a magnetic tape file on a type MTB or MTD controller. On output, ?OPFL/?OPCH contains the target file's channel number.

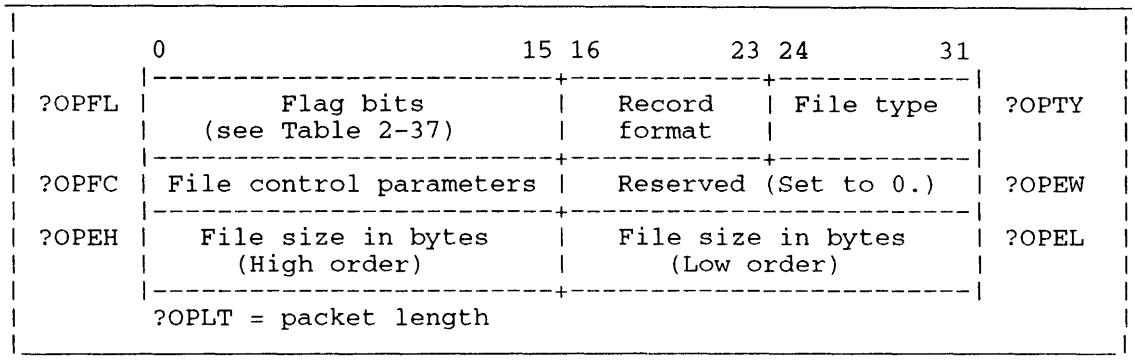


Figure 2–57. Structure of Standard ?GOPEN Packet

Opening System Areas

You can issue ?GOPEN against a system area. To do this, simply place a byte pointer to the desired unit name and system area number in AC0. An example of a system area name is @DPJ0:1082. You must have previously created, via system utility Disk Jockey, the system area. You can open a system area only under the New File System.

Issuing ?GOPEN against a system area is much the same as issuing ?GOPEN against a physical disk alone, but in the first case you can gain access to only part of the disk. Bad block remapping does not occur for system areas that you issue ?GOPEN against.

?GOPEN Options

To open the file to the calling process exclusively, supply value ?OPME in offsets ?OPFL/?OPCH at input. Note that this option is available only within the standard packet (for non–IPC files), because it makes no sense to exclusively open an IPC file.

If you are reading or writing a magnetic tape file on an MTB or MTD tape controller, you can select a density mode for the tape controller by supplying one of the density–mode values in offsets ?OPFL/?OPCH.

Supply ?OPDL to specify a density of 800 bpi (bytes per inch); supply ?OPDM to set the density to 1600 bpi; supply ?OPDH to specify 6250 bpi; or supply ?OPAM if you want the operating system to select the density mode automatically.

Supply ?OPD5, ?OPD6, and ?OPD7 for low, medium, or high densities respectively. When you specify low or high density, the OS selects the lowest or highest density supported by the drive. When you select medium density: if the drive supports three levels, the OS selects the middle density value; when the drive supports two densities, the OS selects the lower density.

If you have issued ?GOPEN with value ?ODTL supplied, you cannot issue the following system calls to the file:

- ?ALLOCATE
- ?BLKIO (write block only)
- ?CPMAX
- ?CRUDA
- ?DELETE
- ?ESFF
- ?GTRUNCATE
- ?PWRB
- ?SACL
- ?UPDATE
- ?WRB
- ?WRUDA

An attempt to issue one of these system calls with value ?ODTL supplied in the ?GOPEN packet results in error code ER_FS_TLA_MODIFY_VIOLATION.

When you default the density–mode parameter (that is, omit the values), the operating system uses the density mode you chose for the MTB or MTD controller during the system–generation procedure. Make sure the default density mode matches that of your tape; otherwise, the operating system fails on error code ERFTM (file tape density mismatch). However, density matching always occurs for reads on MTD drives unless you specify a density.

If you select ?OPAM (automatic density selection) and the tape unit is damaged or the operating system cannot set the density, ?GOPEN fails on error code ERITD (tape density indecipherable).

In buffered mode (Model 6352 magnetic tape units only) the tape controller indicates that a tape transfer is complete after data has been read from memory, but before it has been written to tape. In this mode the system might not report error conditions for a request that fails. Therefore your program must check for errors whenever it issues a ?GCLOSE call. (The system reports all errors when it writes a file mark. Since ?GCLOSE writes a file mark, any program using buffered mode must explicitly close each tape file.)

The streaming mode (Model 6352 magnetic tape units only) allows your program to open a tape unit for high–speed backup purposes. In this mode the tape controller expects data transfer with the tape to occur quite rapidly. If data is not transferred fast enough, the performance of the tape unit degrades. So, a program that cannot maintain a high data transfer rate should not select this mode.

?GOPEN Extension

The ?GOPEN extension is used to allow access to options that exist for some Data General disk subsystems. These options (described below) may not be supported by all disk subsystems. An error will be returned (typically ERIOD or ERMNS) if the option selected is not available for the specified device.

The extension allows access to the following hardware features:

- Format selection for 4514 (DPM) disks.
- Modified sector I/O for DPJ disks (only large capacity models).
- Trespass option for dual ported disks.
- Model option for some MTJ tapes (Model 6352 only).
- Tape drives supporting data compression and the SCSI–2 protocol.

Supply value ?ODTL in offset ?ODF1 of the packet extension when you do not want the operating system to change the file's date/time last accessed values. These values are offsets ?STAH and ?STAL of the parameter packet for system call ?FSTAT and are also the values that the /TLA switch of the CLI FILESTATUS command returns. This change of date/time values occurs by default (i.e., ?ODTL not supplied) when your process later reads or closes the file; it will not occur if you have supplied ?ODTL.

Data compression is on (the default condition) for tape drives supporting compression. You must switch it off if you do not want to use it. Set ?ODCOF to 1 in offset ?ODF1 of the packet extension when you want to turn compression off and operate the drive in native mode.

To use the extension, you must set the extension bit (?OPXP) in the flag word (?OPFL) of the ?GOPEN packet. Figure 2–58 shows the structure of the ?GOPEN packet extension and Table 2–38 contains the option flags for offset ?ODF1 of the ?GOPEN packet extension. The number of words in the combined ?GOPEN main packet and its extension is ?OPLT + ?OPXL.

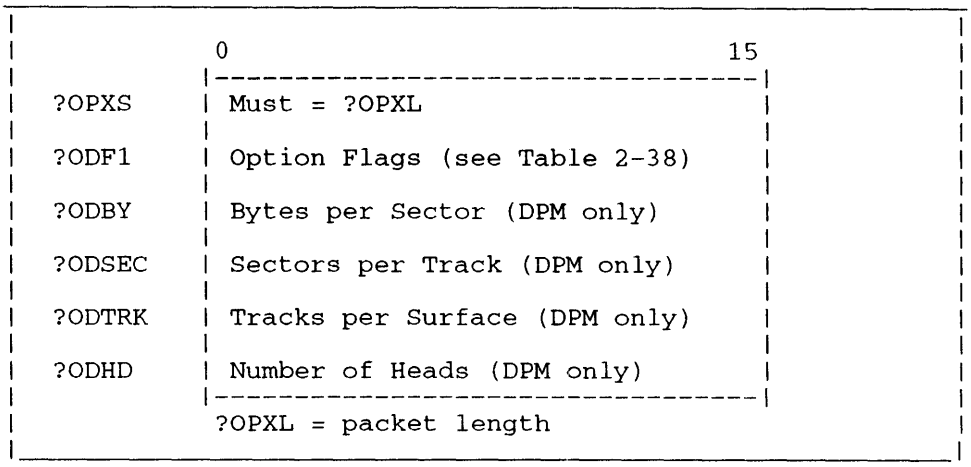


Figure 2–58. Structure of ?GOPEN Packet Extension

Table 2–38. Option Flags for Offset ?ODF1

Flag	Description
?ODND	Write without Modified bits (DPJ only)
?ODMB	Allow Modified Sector I/O (DPJ only)
?ODHS	Swap Head Option (DPM only)
?ODBS	Swap Byte Option (DPM only)
?ODP0	First Physical Sector is 0 (DPM only) (Used for Enterprise/MPT format only)
?ODST	Choose streaming mode for tape (MTJ only)
?ODTEO	9-track tape emulation override
?ODCOF	Data compression off If ?ODCOF = 0 the tape operates in data compression mode. If ?ODCOF = 1, then data compression is turned off, and the drive operates in native mode.
?ODTP	Old file system -- trespass if reserved by another port (DPJ only); new file system -- trespass if reserved by another port or if marked as owned by another system
?ODTL	Do not change the time last accessed value when the file is read from or closed.

Table 2–39 contains the specific format selectors that Figure 2–58 introduces.

Table 2–39. Valid Format Options (DPM disks)

Offset	8 Sector Format	9 Sector Format (Standard AOS)	10 Sector Format (Enterprise/MPT)
?OPXS	?OPXL	?OPXL	?OPXL
?ODF1	0	0	?ODHS+?ODBS+?ODP0
?ODBY	512.	512.	512.
?ODSEC	8.	9.	10.
?ODTRK	40.	40.	35.
?ODHD	1 or 2	1 or 2	2

In order to perform modified sector I/O (DPJ disks) using the ?BLKIO call, bit ?ODMB must be set. If this bit is not set, an error (ERCPO) will be returned. Use the ?ODND option to clear modified bits in conjunction with the ?BLKIO call.

Sample Packet

The following sample packet shows the standard ?GOPEN packet:

```
PKT:    .BLK    ?OPLT           ;Allocate enough space for the
                                ;packet.  Packet length = ?OPLT
        .LOC    PKT+?OPFL       ;Channel number.
        .WORD   0               ;The OS returns this value.
        .LOC    PKT+?OPTY       ;Record format and file type.
        .WORD   0               ;The OS returns this value.
        .LOC    PKT+?OPFC       ;Record length (if fixed format).
        .WORD   0               ;The OS returns this value.
        .LOC    PKT+?OPEW       ;Reserved.
        .DWORD  0               ;You must set this value to 0.
        .LOC    PKT+?OPEH       ;File size (in bytes).
        .DWORD  0               ;The OS returns this value.
        .LOC    PKT+?OPLT       ;End of packet.
```

Notes

- See the description of ?GCLOSE in this chapter.
- Refer to the chapter about VSGEN in the manual *Installing, Starting, and Stopping AOS/VS II* for information on the system–generation procedure.

?GPID *[packet address]*
error return
normal return

Input		Output	
AC0	One of the following: • Host ID • –1 for the current host • Byte pointer to the hostname	AC0	Unchanged
AC1	Defines contents of AC0 as follows: • 0 if AC0 contains a host ID • –1 if AC0 contains a byte pointer • ignored if AC0 contains –1	AC1	Unchanged
AC2	Address of the ?GPID packet, unless you specify the address as an argument to ?GPID	AC2	Address of the ?GPID packet

Error Codes in AC0

ERHNE	Host does not exist
ERPRE	(AC1 = 0 and AC0 <> host ID) or (AC1 = 0 and AC0 <> –1) or (AC1 = –1 and AC0 doesn't contain a byte pointer)
ERVBP	Invalid byte pointer passed as a system call argument
ERVWP	Invalid word pointer passed as a system call argument

Why Use It?

Use this system call to learn the PID numbers of all processes in use on a particular host. This knowledge is useful when you need to address all active processes on a host, such as when you implement functions similar to those of the ?.CLI macro and the Process Environment Display (PED) utility program.

Who Can Use It?

There are no special process privileges needed to issue this call, and there are no restrictions concerning file access.

What It Does

?GPID returns all (or at least as many as will fit in the buffer provided) PIDs that are active on the specified host.

Figure 2–59 shows the structure of ?GPID's parameter packet, and Table 2–40 describes its contents.

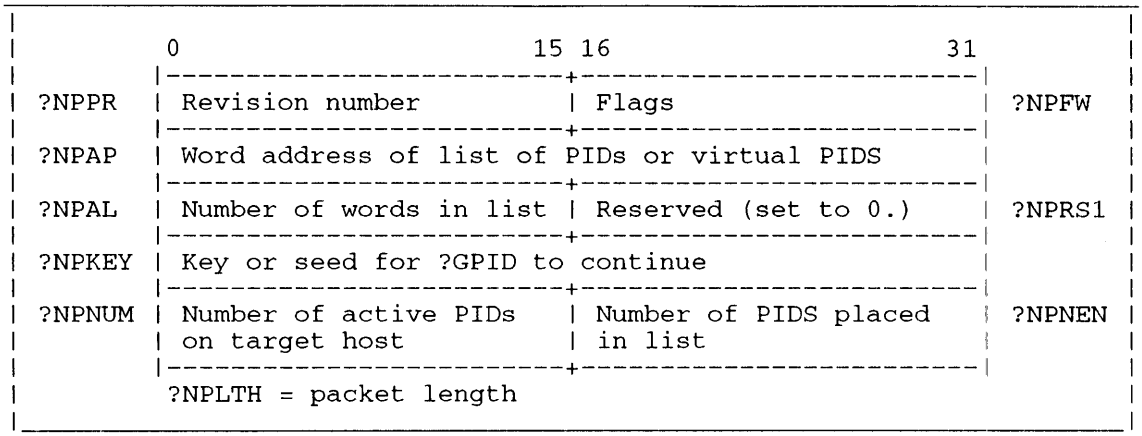


Figure 2–59. Structure of ?GPID Packet

Table 2–40. Contents of ?GPID Packet

Offset	Contents
?NPPR	Packet revision number. Place zero here.
?NPFW	Flags. Set all unused bits to zero. Since no bit offsets are currently defined, place zero here.
?NPAP (double-word)	Word address of an array in which the OS will return the list of PIDs for active processes on the specified host. The list elements are either PIDs or virtual PIDs. Each entry in the array is one word wide.
?NPAL	Length in words of the array that receives the list of PIDs. ?GPID will not return more PIDs than the array will hold. The OS treats ?NPAL as an unsigned number.
?NPRS1	Reserved. (Set to 0.)
?NPKEY (double-word)	Set to 0 the first time you issue this call. If you cannot allocate a large enough array to contain all the PIDs on the specified host, you must reissue the call to obtain the remaining PIDs. In this case, the prior issue of ?GPID has placed a number different from 0 in this offset; leave it here for all reissues. You know when you've obtained all the PIDs (and don't have to reissue ?GPID) when the rest of the buffer contains zeros or when ?NPNEN is less than the array length.
?NPNUM	The OS returns the number of PIDS that were active on the target host when ?GPID executed. You can use this number to adjust the size of the array. The OS always returns ?NPNUM, even if ?NPAL contains zero.
?NPNEN	The OS returns the number of PIDs that it stored in the array. A program that uses this number doesn't have to count the number of unchanged entries to determine how many PIDs were returned. It's your responsibility to initialize the array. It's your responsibility to initialize the array. You can use the ?PIDS system call to find how many PIDs were generated and how many were active; then, using these values, determine the array size you want.

?GPORT Returns the PID associated with a global port number.

?GPORT
error return
normal return

Input

AC0	Reserved (Set to 0.)
AC1	32–bit global port number
AC2	Reserved (Set to 0.)

Output

AC0	Undefined
AC1	PID or virtual PID associated with the global port number
AC2	Local port number that corresponds to the global port number in AC1

Error Codes in AC0

No error codes are currently defined.

Why Use It?

?GPORT allows you to decode a global port number to its local equivalent and to obtain the PID (or virtual PID) of the port's owner. This information can be useful if you are sending or receiving IPC messages on multiple ports.

Who Can Use It?

There are no special process privileges needed to issue this call, and there are no restrictions concerning file access.

What It Does

?GPORT returns the PID (or virtual PID) and local port number associated with the global port number you specify in AC1. (Note that ?GPORT does not check the validity of the global port number you specify.)

?GPOS *[packet address]*
error return
normal return

Input

AC0	Reserved (Set to 0.)
AC1	Reserved (Set to 0.)
AC2	Address of the ?OPEN, ?READ, ?WRITE packet, unless you specify the address as an argument to ?GPOS

Output

AC0	Undefined
AC1	Undefined
AC2	Address of the target file's I/O packet

Error Codes in AC0

ERICN	Illegal channel number
ERIFT	Illegal file type
ERFNO	Channel not open
ERVWP	Invalid word pointer passed as a system call argument

Why Use It?

You can use ?GPOS as a complement to ?SPOS, which repositions the file pointer, or as a complement to the pointer specification you set in the ?READ or ?WRITE packets. You can use a ?GPOS and ?SPOS sequence to perform conditional I/O on a file; for example, to check the position of the file pointer, and then reposition the pointer to skip over certain records in the file.

Using ?GPOS with the pointer specifications in the ?READ or ?WRITE packet gives you similar control over file I/O; that is, you might find the position of the file pointer with ?GPOS, and then adjust the ?READ or ?WRITE packets to read from the beginning of the file or write to the end of the file.

Who Can Use It?

There are no special process privileges needed to issue this call. You needed Execute and Read access to the file's directory, and Read access to the file itself, when you issued ?OPEN against the file.

What It Does

?GPOS returns the current position of the target file's file pointer to offset ?IRNH in the target file's current ?OPEN, ?READ, or ?WRITE packet. (See Figure 2–60.) The operating system expresses the file pointer position (absolute byte number starting with 0 — *not* a record number) as a double–precision integer in this offset. Do not issue ?GPOS against a labeled magnetic tape file.

You can cite the address of the I/O packet as an argument to ?GPOS, or you can load the packet address into AC2 before you issue ?GPOS.

If the operating system returns 0 to offset ?IRNH and the target file is not a fixed–length file, the pointer is set to the first byte (character) in the file. Similarly, 1 in ?IRNH means the file pointer is set to the second byte in the file.

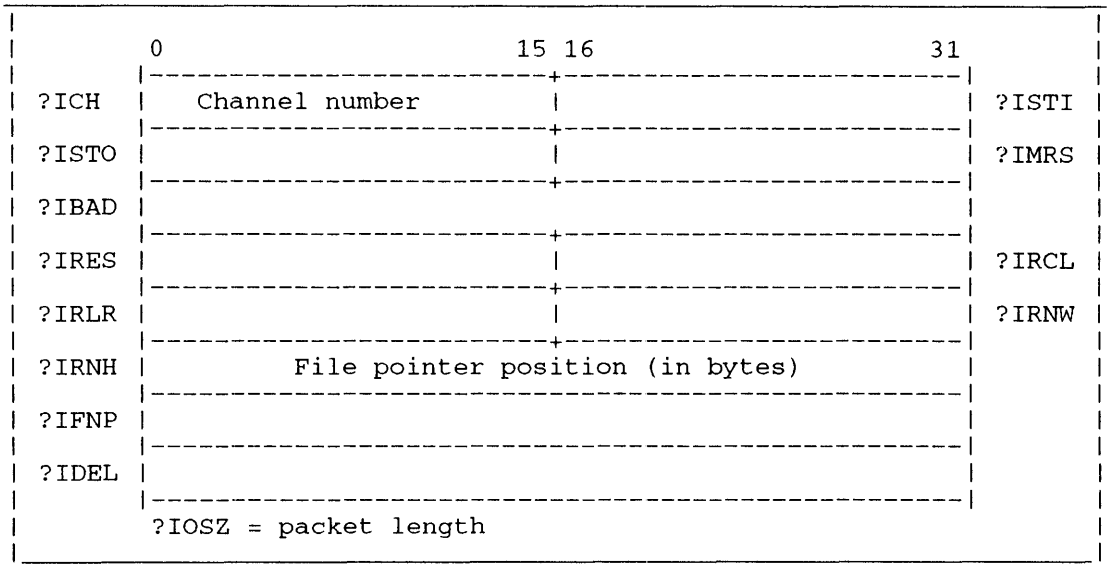


Figure 2–60. Structure of ?GPOS Packet

Notes

- See the descriptions of ?SPOS and ?READ in this chapter for information on setting the file–pointer position.

?GPRNM
error return
normal return

Input

AC0	One of the following: • PID of the target process • –1 to get the pathname of the calling process's program
AC1	Reserved (Set to 0.)
AC2	Byte pointer to a 256–byte buffer

Output

AC0	Unchanged
AC1	Undefined
AC2	Unchanged

Error Codes in AC0

ERPRH	Attempt to access process not in hierarchy
ERVBP	Invalid byte pointer passed as a system call argument

Why Use It?

You can use ?GPRNM to get the pathname of the calling process's program or the pathname of another process.

Who Can Use It?

There are no special process privileges needed to issue this call, and there are no restrictions concerning file access.

What It Does

?GPRNM returns the complete pathname (starting at the root) of the disk file that was originally loaded into the target process's logical address space.

Before you issue ?GPRNM, perform the following steps:

1. Load AC0 with the PID of the target process, or with –1 to get the program name of the calling process.
2. Load AC2 with a byte pointer to a receive buffer you have set aside for the pathname.

AOS/VS

?GRAPHICS *[packet address]*
error return
normal return

Input

AC0	Reserved (Set to 0.)
AC1	Reserved (Set to 0.)
AC2	Address of the ?GRAPHICS packet, unless you specify the address as an argument to ?GRAPHICS.

Output

AC0	Unchanged
AC1	Unchanged
AC2	Address of the ?GRAPHICS packet

Error Codes Returned in AC0

ERIFD	Illegal function for device
ERIGP	Invalid graphics parameter
ERIWR	Invalid window reference
ERPKT	Invalid packet ID
ERRVN	A reserved value was not set to 0
ERVBP	Invalid byte pointer passed as a system call argument
ERVWP	Invalid address passed as a system call argument
ERWNE	The window you specified does not exist

Error codes from the file system

Why Use It?

?GRAPHICS lets you manipulate pixel maps. You can use ?GRAPHICS to create, open, and close a pixel map. You can also use ?GRAPHICS to manipulate the palette associated with a pixel map, to copy a pixel map in and out of your program's address space, and to set a clip rectangle in a pixel map.

Who Can Use It?

There are no special process privileges needed to issue this call, and there are no restrictions concerning file access.

What It Does

?GRAPHICS is a multifunctional system call. To perform a particular function, you set the offset ?GRAPH_PKT.FUNC (in the main ?GRAPHICS packet) to the function code you want. Table 2–41 lists the valid function codes and the functions they perform.

?GRAPHICS is useful only if your program will run on a graphics terminal in an AOS/VS windowing environment. The call applies to one graphics window or pixel map at a time. For example, to find out the status of every pixel map belonging to your program, you must issue the ?GRAPHICS call repeatedly, specifying a different pixel map each time.

Table 2–41. ?GRAPHICS Function Codes

Function Code	What It Lets You Do
?GRAPH_OPEN_WINDOW_PIXELMAP	Open a graphics window's pixel map for graphics output. (See the section "Opening a Graphics Window's Pixel Map.")
?GRAPH_CREATE_MEMORY_PIXELMAP	Create a pixel map in memory, and open it for graphics output. (See the section "Creating a Pixel Map in Memory.")
?GRAPH_CLOSE_PIXELMAP	Close a pixel map you previously opened using ?GRAPH_CREATE_MEMORY_PIXELMAP or ?GRAPH_OPEN_WINDOW_PIXELMAP. (See the section "Closing a Pixel Map.")
?GRAPH_PIXELMAP_STATUS	Get status of a pixel map. (See the section "Getting the Status of a Pixel Map.")
?GRAPH_SET_CLIP_RECTANGLE	Define a clip rectangle on a pixel map; enable or disable the clip rectangle. (See the section "Setting a Clip Rectangle.")
?GRAPH_MAP_PIXELMAP	Copy a pixel map into your program's address space so that you can then use standard MV instructions on it. (See the section "Transferring Data from Pixel Maps to Disk Files.")
?GRAPH_UNMAP_PIXELMAP	Remove a pixel map from your program's address space, so that you can then use graphics instructions on it. (See the section "Transferring Data from Pixel Maps to Disk Files.")
?GRAPH_WRITE_PALETTE	Define colors in the palette associated with a pixel map. (See the section "Writing to a Palette.")
?GRAPH_READ_PALETTE	Find out what colors are stored in the palette associated with a pixel map. (See the section "Reading a Palette.")
?GRAPH_SET_DRAW_ORIGIN	Set the position of the origin used for drawing operations. (See the section "Setting the Draw Origin".)
?GRAPH_GET_DRAW_ORIGIN	Get the position of the origin used for drawing operations. (See the section "Getting the Draw Origin".)

The Main ?GRAPHICS Packet

When you issue a ?GRAPHICS call, you set up a main packet (common to all functions of the call) and a subpacket (unique for each function). Most functions require subpackets. The structure of the

main packet appears in Figure 2–61; we cover the subpackets later, when we discuss each function. Table 2–42 details the contents of each offset in the main packet.

The ?GRAPHICS call applies to one pixel map at a time. For most functions, you specify the target pixel map by supplying a pixel map ID in the main packet. (Functions ?GRAPH_OPEN_WINDOW_PIXELMAP and ?GRAPH_CREATE_MEMORY_PIXELMAP are exceptions which we explain later.)

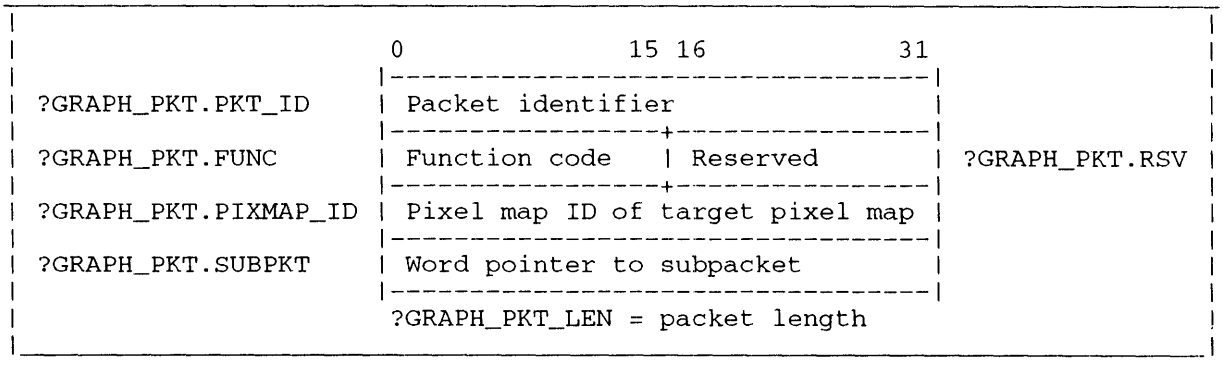


Figure 2–61. Structure of the ?GRAPHICS Main Packet

Table 2–42. Contents of the ?GRAPHICS Main Packet

Offset	Contents
?GRAPH_PKT.PKT_ID (doubleword)	Packet identifier; set to ?GRAPH_PKT_PKTID.
?GRAPH_PKT.FUNC	Code for the function you want. You must always supply a function code. (Function codes are listed in Table 2-41.)
?GRAPH_PKT.RSV	Reserved. (Set to 0.)
?GRAPH_PKT.PIXMAP_ID (doubleword)	Pixel map ID of the target pixel map. You must supply a pixel map ID for all functions except ?GRAPH_CREATE_MEMORY_PIXELMAP and ?GRAPH_OPEN_WINDOW_PIXELMAP. On these functions, the operating system returns a pixel map ID; set this offset to 0.
?GRAPH_PKT.SUBPKT (doubleword)	Word pointer to the subpacket for the function. If the function does not require a subpacket, set this offset to 0.

Opening a Graphics Window's Pixel Map

The virtual terminal associated with a graphics window is a pixel map, which can accept only output in the form of graphics instructions. When you issue graphics instructions, you specify the destination pixel map by a pixel map ID (which you can think of as a "graphics output channel"). A single graphics window can have more than one pixel map ID at a time, since you can open its pixel map as many times as you want.

NOTE: Before you open a graphics window, you must first create it using the ?WIN_CREATE_WINDOW function of ?WINDOW.

To get a pixel map ID for a graphics window, issue the ?GRAPHICS call with function code ?GRAPH_OPEN_WINDOW_PIXELMAP. In the main packet, set offset ?GRAPH_PKT.PIXMAP_ID to 0; the operating system returns the pixel map ID in this field.

In the subpacket, specify the graphics window you want to open. You can specify the window in one of three ways:

- Channel number — When you open a graphics window for input (using ?OPEN), the operating system returns a channel number. (Note that you can use the channel number for input only; you need a pixel map ID to send output to a graphics window.)
- Window pathname — When you create a window, you give it a window name. A window's pathname takes the form @PMAPn:windowname, where @PMAPn is the name of the pixel–mapped terminal that contains the window. The window pathname must end in a null character, <0>; it can be a maximum of ?MXPL characters long (including the null terminator).
- Window ID number — When you create a window, the operating system returns the window ID number in the main ?WINDOW packet.

Fill in the appropriate offset in the subpacket (channel number, pathname or ID number), and set the other two window specification offsets to zero. To indicate which method you are using to specify the window, you must set the appropriate flag in flag word ?GRAPH_OPEN.FLAGS. You can use only one method at a time.

Figure 2–62 shows the subpacket structure; Table 2–43 details the contents of the subpacket.

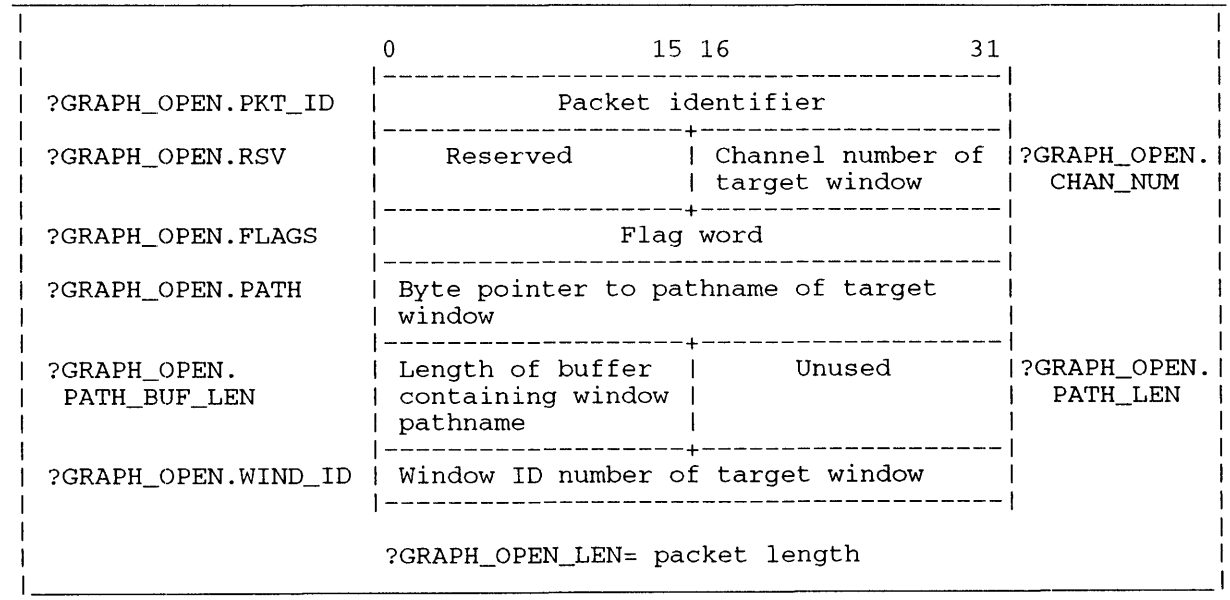


Figure 2–62. Structure of the ?GRAPH_OPEN_WINDOW_PIXELMAP Subpacket

Table 2–43. Contents of the ?GRAPH_OPEN_WINDOW_PIXELMAP Subpacket

Offset	Contents
?GRAPH_OPEN.PKT_ID (doubleword)	Packet identifier; set to ?GRAPH_OPEN_PKTID.
?GRAPH_OPEN.RSV	Reserved. (Set to 0.)
?GRAPH_OPEN.CHAN_NUM	Channel number of target graphics window. When specifying the window by its channel number, you must also set flag ?GRAPH_OPEN.FLAGS.IN_CHAN in flag word ?GRAPH_OPEN.FLAGS. Set this offset to 0 when specifying the target window by its pathname or window ID.
?GRAPH_OPEN.FLAGS (doubleword)	Flag word ?GRAPH_OPEN.FLAGS.IN_PATH -- Set this flag when specifying the target window by its pathname. ?GRAPH_OPEN.FLAGS.IN_WIND_ID -- Set this flag when specifying the target window by its window ID. ?GRAPH_OPEN.FLAGS.IN_CHAN -- Set this flag when specifying the target window by its channel number.
?GRAPH_OPEN.PATH (doubleword)	Byte pointer to pathname of target window. When specifying the window by its pathname, you must also set flag ?GRAPH_OPEN.FLAGS.IN_PATH in flag word ?GRAPH_OPEN.FLAGS. Set this offset to 0 when specifying the target window by its channel number or window ID.
?GRAPH_OPEN.PATH_BUF_LEN	Length of the buffer containing the window pathname. The buffer length must include the null terminator; the maximum buffer length is ?MXPL.
?GRAPH_OPEN.PATH_LEN	Unused. (Set to 0.)
?GRAPH_OPEN.WIND_ID (doubleword)	Window ID number of target window. When specifying the window by its window ID, you must also set flag ?GRAPH_OPEN.FLAGS.IN_WIND_ID in flag word ?GRAPH_OPEN.FLAGS. Set this offset to 0 when specifying the target window by its pathname or channel number.

Creating a Pixel Map in Memory

You use the ?GRAPH_CREATE_MEMORY_PIXELMAP function to create a pixel map in memory,

and to open it for graphics output. Although such a pixel map is not visible on the physical screen, you can still perform graphics instructions on it, copy its contents to a graphics window's pixel map, and perform ?GRAPHICS functions on it.

In the main packet, set offset ?GRAPH_PKT.PIXMAP_ID to 0; the operating system returns a pixel map ID when it opens the newly created pixel map.

Your entries in the subpacket determine the attributes (width, height and pixel depth) of the new pixel map. Figure 2–63 shows the subpacket structure; Table 2–44 details its contents.

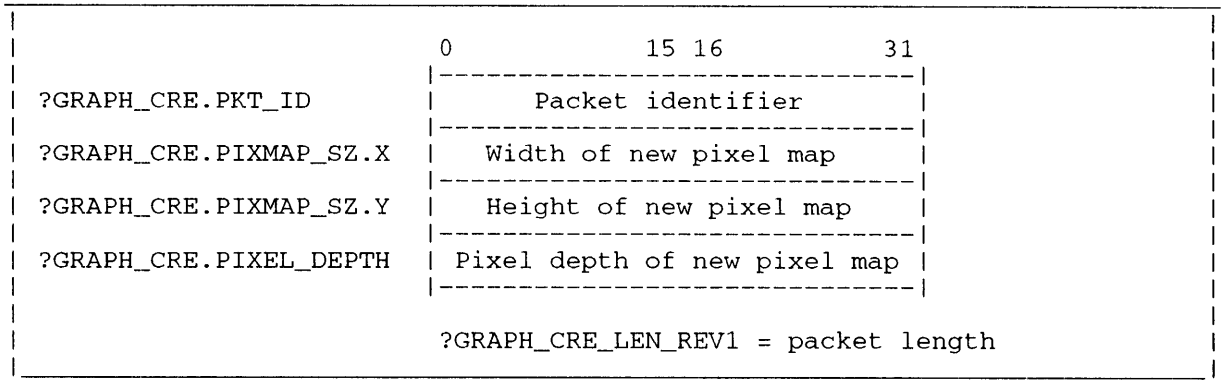


Figure 2–63. Structure of the ?GRAPHICS_CREATE_MEMORY_PIXELMAP Subpacket

Table 2–44. Contents of the ?GRAPH_CREATE_MEMORY_PIXELMAP Subpacket

Offset	Contents
?GRAPH_CRE.PTK_ID (doubleword)	Packet identifier. Set to ?GRAPH_CRE_PKTID_REV1.
?GRAPH_CRE.PIXMAP_SZ.X (doubleword)	Width of pixel map (in pixels).
?GRAPH_CRE.PIXMAP_SZ.Y (doubleword)	Height of pixel map (in pixels).
?GRAPH_CRE.PIXEL_DEPTH (doubleword)	Pixel depth of pixel map (in bits).

Closing a Pixel Map

You use the ?GRAPHICS function ?GRAPH_CLOSE_PIXELMAP to close an open pixel map. The open pixel map can be a memory pixel map, or the pixel map associated with a graphics window.

All memory pixel maps are open: when you create a memory pixel map, the operating system automatically opens it. When you close a memory pixel map, the operating system automatically deletes it.

A graphics window's pixel map is open only if you have issued the ?GRAPH_OPEN_WINDOW_PIXELMAP function for that window. You can open a graphics window's pixel map more than once; each time, the operating system returns a separate pixel map ID. When you close a graphics window's pixel map, the operating system closes the pixel map ID you specified. If you open a graphics window's pixel map more than once, you must issue a separate ?GRAPH_CLOSE_PIXELMAP call for each pixel map ID. (To delete a graphics window's pixel map, you must delete the window itself; use the ?WINDOW call, function code ?WIN_DELETE_WINDOW.)

In the main packet, specify the pixel map ID of the pixel map you want to close. Since there is no subpacket for this function, set ?GRAPH_PKT.SUBPKT to 0.

Getting the Status of a Pixel Map

You use ?GRAPH_PIXELMAP_STATUS to get current information about a pixel map. You can get information about either a memory pixel map or a graphics window's pixel map.

In the main packet, specify the pixel map ID of the pixel map you want. In the subpacket, supply a packet identifier; set all other offsets to 0. The operating system returns the following information in the subpacket:

- Width and height of the pixel map.
- Pixel depth.
- The coordinates of the clip rectangle's origin.
- Dimensions of the clip rectangle.
- Status of the clip rectangle: enabled or disabled.

Figure 2–64 shows the subpacket structure; Table 2–45 details its contents.

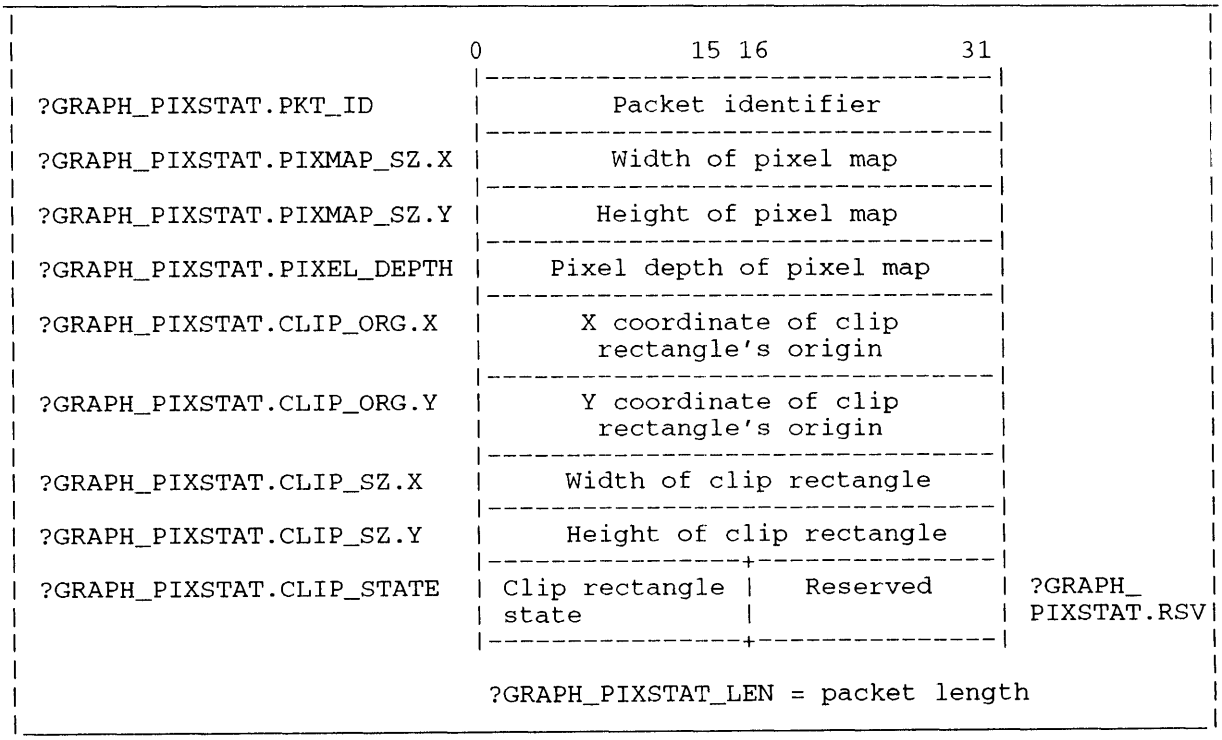


Figure 2–64. Structure of the ?GRAPH_PIXELMAP_STATUS Subpacket

Table 2–45. Contents of the ?GRAPH_PIXELMAP_STATUS Subpacket

Offset	Contents
?GRAPH_PIXSTAT.PKT_ID (doubleword)	Packet identifier. Set to ?GRAPH_PIXSTAT_PKTID.
?GRAPH_PIXSTAT.PIXMAP_SZ.X (doubleword)	Width (in pixels) of pixel map. Returned by the operating system.
?GRAPH_PIXSTAT.PIXMAP_SZ.Y (doubleword)	Height (in pixels) of pixel map. Returned by the operating system.
?GRAPH_PIXSTAT.PIXEL_DEPTH (doubleword)	Pixel depth of pixel map (number of bits per pixel).
?GRAPH_PIXSTAT.CLIP_ORG.X (doubleword)	X coordinate of clip rectangle's origin (in pixels relative to origin of pixel map).
?GRAPH_PIXSTAT.CLIP_ORG.Y (doubleword)	Y coordinate of clip rectangle's origin (in pixels relative to origin of pixel map).
?GRAPH_PIXSTAT.CLIP_SZ.X (doubleword)	Width (in pixels) of clip rectangle.
?GRAPH_PIXSTAT.CLIP_SZ.Y (doubleword)	Height (in pixels) of clip rectangle.
?GRAPH_PIXSTAT.CLIP_STATE	The current state of the clip rectangle. ?GRAPH_RECT_STATE_ENABLE -- Indicates that the clip rectangle is currently enabled. ?GRAPH_RECT_STATE_DISABLE -- Indicates that the clip rectangle is currently disabled.
?GRAPH_PIXSTAT.RSV	Reserved. (Set to 0.)

Setting the Clip Rectangle

You use the ?GRAPH_SET_CLIP_RECTANGLE function to specify a rectangular portion of a pixel map as a clip rectangle. When a clip rectangle is in effect, you can still specify any location on the pixel map as an argument to a graphics instruction. However, any changes affect only the portion of the pixel map within the rectangle.

Every pixel map has a clip rectangle. Initially, the clip rectangle has the same size and origin as the pixel map, and clipping is disabled. The ?GRAPH_SET_CLIP_RECTANGLE function lets you set the clip rectangle's origin, width and height, and lets you disable or re–enable clipping.

In the main packet, specify the pixel map ID of the pixel map for which you want to set the clip rectangle.

Your entries in the subpacket determine which attributes of the clip rectangle (its size, origin, or state) will change. To reset a particular attribute, place the new value in the appropriate offset; in the flag word ?GRAPH_SET_CLIP.FLAGS, set the flag that corresponds to that offset. If you don't want to change a particular attribute, set both the offset and the corresponding flag to 0.
Figure 2–65 shows the subpacket structure; Table 2–46 details its contents.

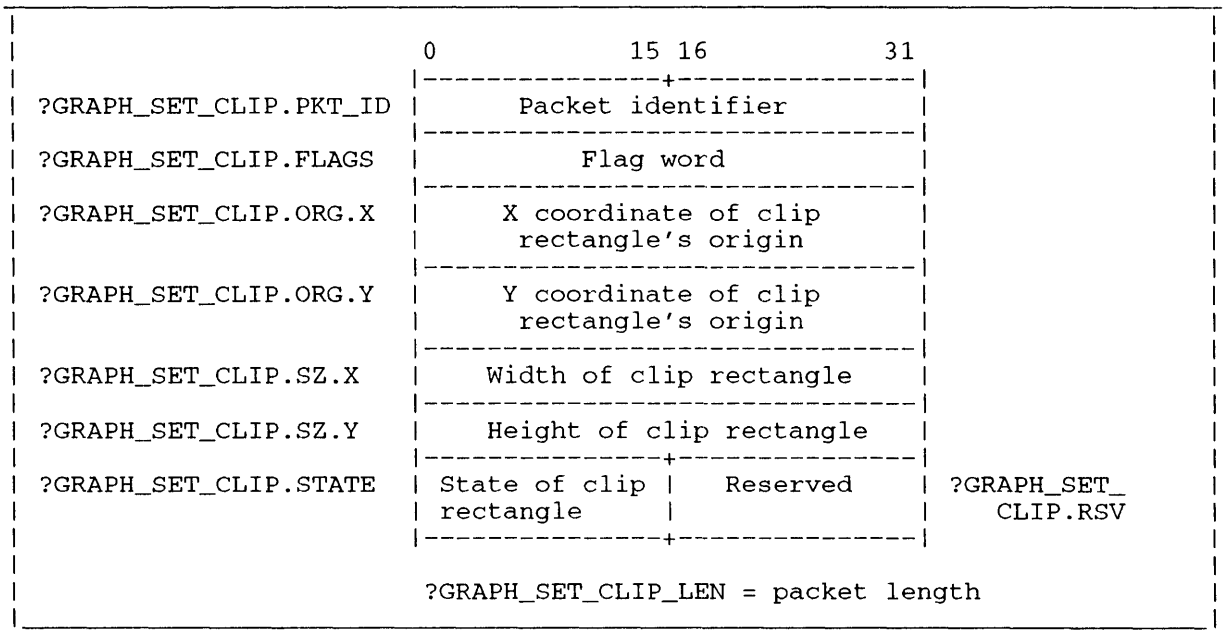


Figure 2–65. Structure of the ?GRAPH_SET_CLIP_RECTANGLE Subpacket

Table 2–46. Contents of the ?GRAPH_SET_CLIP.RECTANGLE Subpacket

Offset	Contents
?GRAPH_SET_CLIP.PKT_ID (doubleword)	Packet identifier. Set this offset to ?GRAPH_SET_CLIP_PKTID.
?GRAPH_SET_CLIP.FLAGS (doubleword)	Flag word. For each clip rectangle attribute you want to change, you must set the corrresponding flag. ?GRAPH_SET_CLIP.FLAGS.ORG_X -- You must set this flag if you are changing the X coordinate of the clip rectangle's origin. ?GRAPH_SET_CLIP.FLAGS.ORG_Y -- You must set this flag if you are changing the Y coordinate of the clip rectangle's origin. ?GRAPH_SET_CLIP.FLAGS.SZ_X -- You must set this flag if you are changing the width of the clip rectangle. ?GRAPH_SET_CLIP.FLAGS.SZ_Y -- You must set this flag if you are changing the height of the clip rectangle. ?GRAPH_SET_CLIP.FLAGS.STATE -- You must set this flag if you are changing the state of the clip rectangle.

(continued)

Table 2–46. Contents of the ?GRAPH_SET_CLIP.RECTANGLE Subpacket

Offset	Contents
?GRAPH_SET_CLIP.ORG.X (doubleword)	X coordinate of clip rectangle's origin (in pixels, relative to origin of pixel map). If you supply a value here, you must also set the flag ?GRAPH_SET_CLIP.FLAGS.ORG_X. If you are not changing the X coordinate, set this offset to 0, and do not set the corresponding flag.
?GRAPH_SET_CLIP.ORG.Y (doubleword)	Y coordinate of clip rectangle's origin (in pixels, relative to origin of pixel map). If you supply a value here, you must also set the flag ?GRAPH_SET_CLIP.FLAGS.ORG_Y. If you are not changing the Y coordinate, set this offset to 0, and do not set the corresponding flag.
?GRAPH_SET_CLIP.SZ.X (doubleword)	Width of clip rectangle (in pixels). If you supply a value here, you must also set the flag ?GRAPH_SET_CLIP.FLAGS.SZ.X. If you are not changing the width, set this offset to 0, and do not set the corresponding flag.
?GRAPH_SET_CLIP.SZ.Y (doubleword)	Height of clip rectangle (in pixels). If you supply a value here, you must also set the flag ?GRAPH_SET_CLIP.FLAGS.SZ.Y. If you are not changing the width, set this offset to 0, and do not set the corresponding flag.
?GRAPH_SET_CLIP.STATE	State of clip rectangle. To change the state, place one of the following values here, and set the flag ?GRAPH_SET_CLIP.FLAGS.STATE. If you are not changing the state, set this offset to 0, and do not set the corresponding flag. ?GRAPH_RECT_STATE_ENABLE -- Enables the clip rectangle. ?GRAPH_RECT_STATE_DISABLE -- Disables the clip rectangle.
?GRAPH_SET_CLIP.RSV	Reserved. (Set to 0.)

(concluded)

Transferring Data Between Pixel Maps and Disk Files

To transfer information between a disk file and a pixel map,

1. Issue the ?GRAPHICS call with function code ?GRAPH_MAP_PIXELMAP to map an existing memory pixel map into your program's address space. (We explain this step in more detail in the section "To Map a Pixel Map Into Your Program's Address Space" below.)
2. Use I/O system calls, such as ?READ and ?WRITE, to transfer data between the disk file and the address space.
3. Issue ?GRAPHICS with function code ?GRAPH_UNMAP_PIXELMAP to unmap the pixel map from the address space. This returns the contents of the address space to pixel map format; you can then use graphics instructions on that pixel map. (We explain this step in more detail in the section "To Unmap a Pixel Map from Your Program's Address Space" below.)

To Map a Pixel Map into Your Program's Address Space

You use the ?GRAPH_MAP_PIXELMAP function to load a pixel map into your program's address space. In the main packet, specify the pixel map ID of the pixel map you want.

The operating system calculates where to place the pixel map in your address space, and returns the address of the pixel map in the subpacket (offset ?GRAPH_MAP_PMAP.ADDR). It also returns information that lets you convert pixel map coordinates into word and bit addresses so that you can perform data transfer instructions accurately.

Figure 2–66 shows the subpacket structure; Table 2–47 details its contents.

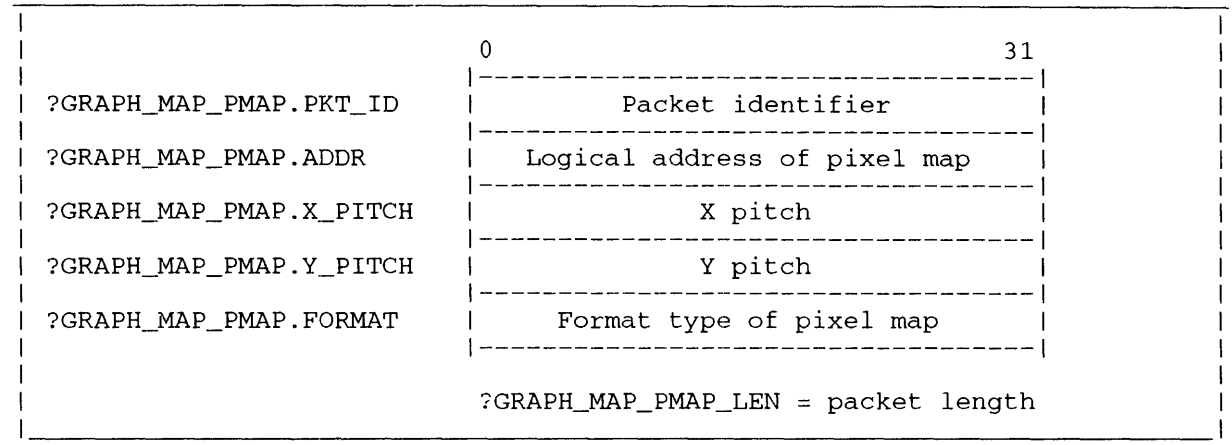


Figure 2–66. Structure of the ?GRAPH_MAP_PIXELMAP Subpacket

Table 2–47. Contents of the ?GRAPH_MAP_PIXELMAP Subpacket

Offset	Contents
?GRAPH_MAP_PMAP.PKT_ID (doubleword)	Packet identifier. Set to ?GRAPH_MAP_PMAP_PKTID.
?GRAPH_MAP_PMAP.ADDR (doubleword)	Logical address of pixel map in your program's address space (returned by the operating system).
?GRAPH_MAP_PMAP.X_PITCH	The number of bits occupied by a pixel; not necessarily the same as pixel depth. (Returned by the operating system.)
?GRAPH_MAP_PMAP.Y_PITCH	The number of pixels on a line; not necessarily the same as the width of the pixel map. (Returned by the operating system.)
?GRAPH_MAP_PMAP.FORMAT (doubleword)	Pixel map format type (returned by the operating system). ?GRAPH_MAP_PMAP.FORMAT.PACKED_PIXEL -- Standard format for Data General pixel maps.

To get the bit offset (from the beginning of the mapped pixel map) of a particular pixel, use the following formula:

Bit offset = (y_coordinate * X_PITCH * Y_PITCH) + (x_coordinate * X_PITCH)

To Unmap a Pixel Map from Your Program's Address Space

Once you have mapped a pixel map into your program's address space, you cannot use graphics instructions on that pixel map until you have unmapped it.

To unmap a pixel map, issue ?GRAPHICS with function code ?GRAPH_UNMAP_PIXELMAP. The operating system then unmaps that pixel map from your program's address space to the pixel map's original location. Any data you wrote to the pixel map while it was in your address space will appear in the unmapped pixel map, and you can then use graphics instructions on that pixel map.

In the main packet, specify the pixel map ID of the target pixel map. There is no subpacket for the ?GRAPH_UNMAP_PIXELMAP function; set offset ?GRAPH_PKT.SUBPKT to 0.

Writing to a Palette

You use function ?GRAPH_WRITE_PALETTE to change the colors in the palette associated with a pixel map. You can change the color settings for one or more contiguous pixel values. For the set of pixel values you specify, you can change either the phase 0 setting, the phase 1 setting, or both. Follow these steps to change the colors in a palette:

1. Set up an array in memory to contain the color settings. If you are changing both the phase 0 and phase 1 settings, you must set up two separate arrays, one for each phase. The arrays must be the same size.

   There must be an entry in each array for each pixel value you want to change. An entry consists of four 32–bit unsigned fractions, one for each color level:

   <Red level> <Green level> <Blue level> <Grey level>

2. Issue the ?GRAPHICS call with function code ?GRAPH_WRITE_PALETTE. In the main packet, specify the pixel map ID of the pixel map whose palette you are changing. In the subpacket, you specify

   - The number of palette entries you want to write.
   - The palette index at which you want to start writing.
   - The addresses of the arrays you have stored in memory.

Figure 2–67 shows the subpacket structure.

The operating system then copies the array(s) you specify into the palette, starting at the specified palette index.

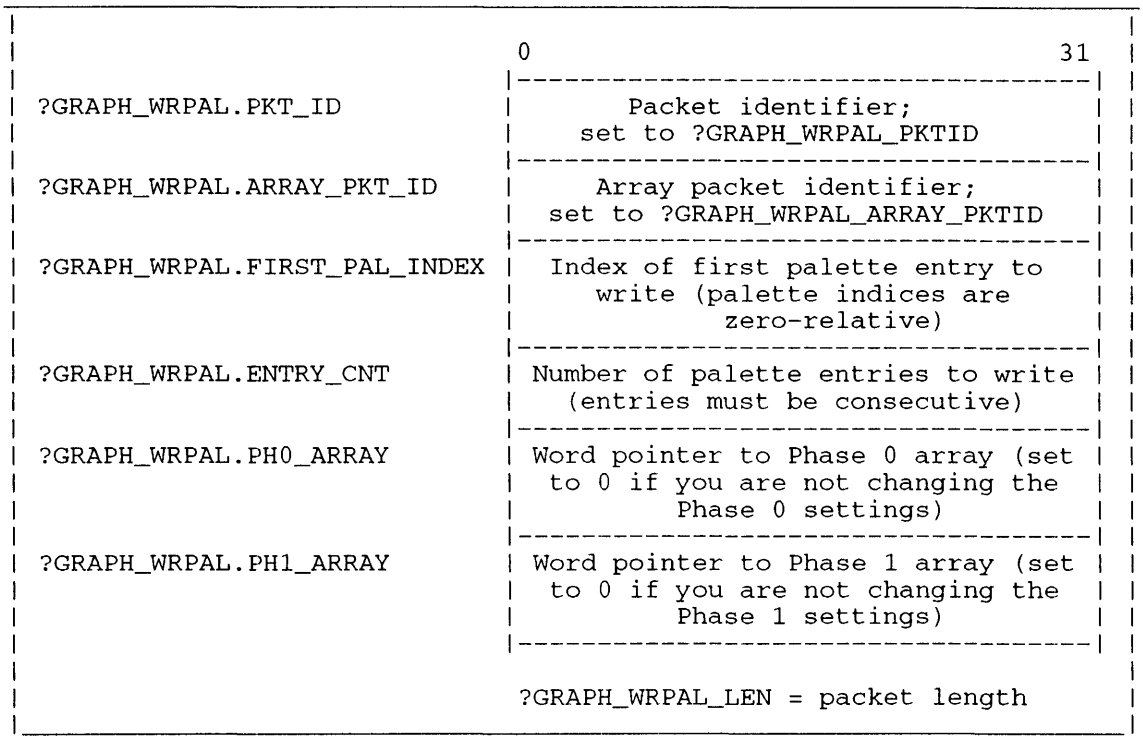


Figure 2-67. Structure of the ?GRAPH_WRITE_PALETTE Subpacket

Reading from a Palette

You can use function ?GRAPH_READ_PALETTE to get the current color settings from the palette associated with a pixel map. This call copies a series of color settings for contiguous pixel values from the palette into memory. Follow these steps to determine the colors in a palette:

1. Allocate space in memory for each array you are copying. If you are copying both the Phase 0 and Phase 1 settings, you must allocate space for two separate arrays (one for each phase). Each array needs as much space as the number of entries you will copy, multiplied by the size of each entry (four doublewords).

2. Issue the ?GRAPHICS call with function code ?GRAPH_READ_PALETTE. In the main packet, specify the pixel map ID of the pixel map whose palette you are reading. In the subpacket, you specify

   - The number of palette entries you are reading.
   - The palette index at which you want to start reading.
   - The destination memory address for each array.

The operating system then copies the palette entries you specified into the space you allocated in memory. The copied entries are set up as arrays, one for each phase. Each array contains an entry for each pixel value you requested. Each entry consists of four doublewords, each of which denotes a color level:

<Red level> <Green level> <Blue level> <Grey level>

Figure 2–68 shows the subpacket structure.

	0 31
?GRAPH_RDPAL.PKT_ID	Packet identifier; set to ?GRAPH_RDPAL_PKTID
?GRAPH_RDPAL.ARRAY_PKT_ID	Array packet identifier; set to ?GRAPH_RDPAL_ARRAY_PKTID
?GRAPH_RDPAL.FIRST_PAL_INDEX	Index of first palette entry to read. Palette indices are zero-relative.
?GRAPH_RDPAL.ENTRY_CNT	Number of palette entries to read (entries must be consecutive)
?GRAPH_RDPAL.PH0_ARRAY	Word pointer to Phase 0 array (set to 0 if you are not reading the Phase 0 settings)
?GRAPH_RDPAL.PH1_ARRAY	Word pointer to Phase 1 array (set to 0 if you are not reading the Phase 1 settings)
	?GRAPH_RDPAL_LEN = packet length

Figure 2-68. Structure of the ?GRAPH_READ_PALETTE Subpacket

Setting the Draw Origin

The origin of a pixel map — the pixel at position (0,0) — is usually at the pixel map's upper–left corner. However, for drawing operations, you can move the origin to facilitate panning. (Moving the draw origin changes only the coordinates you use for drawing; for other operations, such as performing windowing functions, setting the clip rectangle, or working with pointer events, the coordinates remain unchanged.)

To change the location of the draw origin, issue the ?GRAPHICS call with function code ?GRAPHICS_SET_DRAW_ORIGIN. In the main packet, specify the pixel map ID you want.

NOTE: This call affects only the pixel map ID you specify; if a pixel map has additional pixel map IDs, the drawing coordinates for those pixel map IDs will remain unchanged.)

In the subpacket, specify the new position of the draw origin. You can move the origin anywhere you want, either on or off the pixel map. The coordinates are in pixels and are relative to the upper–left corner of the pixel map. For example, to set the draw origin to 100 pixels down and 200 pixels to the right of the upper–left corner of the pixel map, you would specify coordinates of (100, 200). To move the draw origin to the upper–left corner, you would specify (0,0). To move the draw origin 50 pixels above and 30 pixels to the left of the upper–left corner, you would specify (–50,–30).

Figure 2–69 shows the subpacket structure; Table 2–48 details its contents.

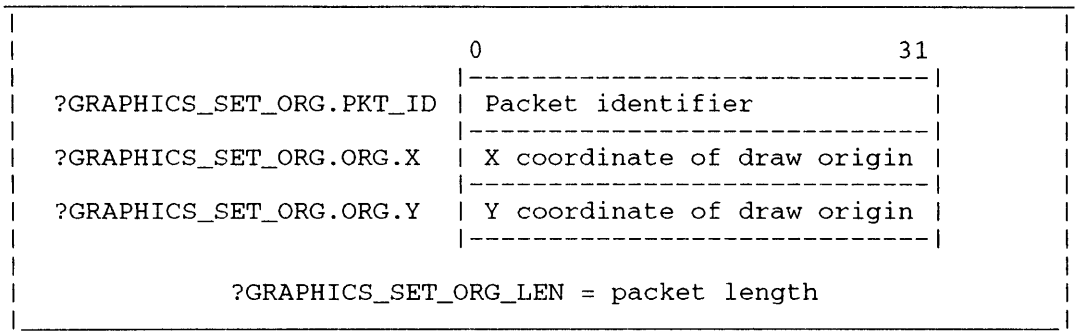


Figure 2–69. Structure of the ?GRAPHICS_SET_DRAW_ORIGIN Subpacket

Table 2–48. Contents of the ?GRAPHICS_SET_DRAW_ORIGIN Subpacket

Offset	Contents
?GRAPHICS_SET_ORG.PKT_ID (doubleword)	Packet identifier; set to ?GRAPHICS_SET_ORG_PKTID.
?GRAPHICS_SET_ORG.ORG.X (doubleword)	New X coordinate of draw origin. Specify the coordinate in pixels, relative to the upper-left corner of the pixel map. (A positive value will place the draw origin to the right of the upper-left corner; a negative value places the origin to the left of the upper-left corner. A value of 0 places the origin on the left boundary of the pixel map.)
?GRAPHICS_SET_ORG.ORG.Y (doubleword)	New Y coordinate of draw origin. Specify the coordinate in pixels, relative to the upper-left corner of the pixel map. (A positive value will place the draw origin below the upper-left corner; a negative value places the origin above the upper-left corner. A value of 0 places the origin on the upper boundary of the pixel map.)

Getting the Coordinates of the Draw Origin

To find out the location of the draw origin for a particular pixel map ID, issue the ?GRAPHICS call with function code ?GRAPHICS_GET_DRAW_ORIGIN. In the main packet, specify the pixel map ID you want.

NOTE: This call returns only the draw origin of the particular pixel map ID you specify. A pixel map can have several pixel map IDs, each with its own independent draw origins; to get the draw origins for all the pixel map IDs, you must reissue this call for each pixel map ID.

The operating system returns the coordinates of the draw origin in the subpacket. The coordinates are in pixels and are relative to the upper–left corner of the pixel map.

Figure 2–70 shows the subpacket structure; Table 2–49 details its contents.

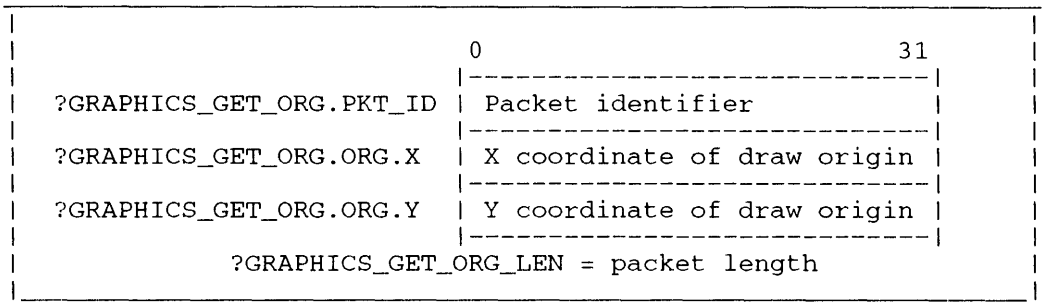


Figure 2–70. Structure of the ?GRAPHICS_GET_DRAW_ORIGIN Subpacket

Table 2–49. Contents of the ?GRAPHICS_GET_DRAW_ORIGIN Subpacket

Offset	Contents
?GRAPHICS_GET_ORG.PKT_ID (doubleword)	Packet identifier; set to ?GRAPHICS_GET_ORG_PKTID.
?GRAPHICS_GET_ORG.ORG.X (doubleword)	X coordinate of draw origin; in pixels, relative to the upper-left corner of the pixel map. (A positive value indicates the origin is to the right of the upper-left corner; a negative value indicates the origin is to the left of the upper-left corner. A value of 0 indicates that the origin is even with the left boundary of the pixel map.)
?GRAPHICS_GET_ORG.ORG.Y (doubleword)	Y coordinate of draw origin; in pixels, relative to the upper-left corner of the pixel map. (A positive value indicates the origin is below the upper-left corner; a negative value indicates the origin is above the upper-left corner. A value of 0 indicates that the origin is even with the upper boundary of the pixel map.

Notes

- See the description of ?GECHR in this chapter.

?GRNAME — Returns complete pathname of generic file.

?GRNAME
error return
normal return

Input		Output	
AC0	Byte pointer to the input generic filename	AC0	Unchanged
AC1	Byte pointer to the buffer	AC1	Unchanged
AC2	Length of the buffer in bytes	AC2	Length of the name in bytes, excluding the null terminator

Error Codes in AC0

ERFDE	File does not exist (The file may be a generic–to–real connection exists.)
ERVBP	Invalid byte pointer passed as a system call argument

Why Use It?

?GRNAME allows you to determine what file is associated with a particular generic file. Also, you can use ?GRNAME to return the complete pathname of a nongeneric pathname fragment in the same way that you would use ?GNAME.

Who Can Use It?

There are no special process privileges needed to issue this call, and there are no restrictions concerning file access.

What It Does

?GRNAME returns the complete pathname associated with a generic file.

If the input file is a generic file and there is no connected "real" file, then the operating system returns error code ERFDE in AC0 and sets Bit 0 of AC2 to 0. If the file is a generic file that is connected to a "real" file, then the operating system returns the full pathname of the connected file as if a ?GNAME were performed on that pathname. In this case, the operating system sets Bit 0 of AC2 to 1.

Notes

- You cannot always use ?GNAME to obtain the "true" pathname of a generic file. For example, ?GNAME would return a complete pathname of :PER:DATA for the input pathname @DATA. However, the complete pathname of the file might actually be :UDD:USER:DATA. In this case, ?GRNAME would return :UDD:USER:DATA.
- If you place in AC0 a byte pointer to the text string @NULL, then ?GRNAME places :PER:NULL in the buffer whose byte pointer is in AC1.
- See the description of ?GNAME in this chapter.

AOS/VS II only

?GROUP *[packet address]*
error return
normal return

Input		Output	
AC0	Reserved (Set to 0)	AC0	Error code
AC1	Unused (Set to 0)	AC1	Undefined
AC2	?GROUP Packet Address (unless specified as a system call parameter)	AC2	Unchanged

Error Codes in AC0

ERPKT Invalid packet ID
ERPRE Invalid system call parameter
ERRVN Reserved value not zero
ERVBP Invalid byte pointer passed as system call argument
ER_FS_GROUPNAME_TOO_LONG
ER_FS_GROUP_ALREADY_EXISTS
ER_FS_GROUP_BUFFER_TOO_SMALL
ER_FS_GROUP_DIR_DOES_NOT_EXIST
ER_FS_GROUP_DOES_NOT_EXIST
ER_FS_GROUP_LIST_INPUT_MISMATCH
ER_FS_ILLEGAL_GROUPNAME_CHARACTER
ER_FS_ILLEGAL_GROUP_LIST_FORMAT
ER_FS_INVALID_GROUP_FUNCTION
ER_FS_INVALID_GROUP_LIST_BYTE_PTR
ER_FS_INVALID_PKT_PTR
ER_FS_TOO_MANY_GROUPS_SPECIFIED
FS_GROUP_COMMENT_END_NO_BEGIN
FS_GROUP_FILE_CONTAINS_UNTERMED_COMMENT
FS_ILLEGAL_CHAR_IN_GROUP_FILE
FS_USERNAME_TOO_LONG_IN_GROUP_FILE

Why Use It?

?PROC initially defines the group access control list of the process. ?GROUP allows you to modify it.

Who Can Use It?

No special privileges are required for a process to obtain or change it's current group access control list. The process must have execute access to the :GROUPS directory to set a group access control list. The process must have execute access to the :GROUPS directory and read access to the group file to view the names in the group file.

What It Does

Use the ?GROUP_GET_LIST function to read the group access control list into the group buffer. Modify the group buffer by adding or removing group names. The group buffer contains a double null–terminated list of group names. A group name is a null–terminated, ASCII string with a maximum length of 16 bytes, including the null character. A group name corresponds to a filename in the :GROUPS directory.

The ?GROUP_REPLACE function sets the group access control list of the process. Before setting the list of group names, the system call reads each group file and verifies that the process–related username is a member of the group.

A group file contains a list of usernames that belong to the group. A group access control list can contain up to ?GROUP_NUM_MAX number of names.

Figure 2–71 shows the format of the packet and Table 2–50 defines the contents.

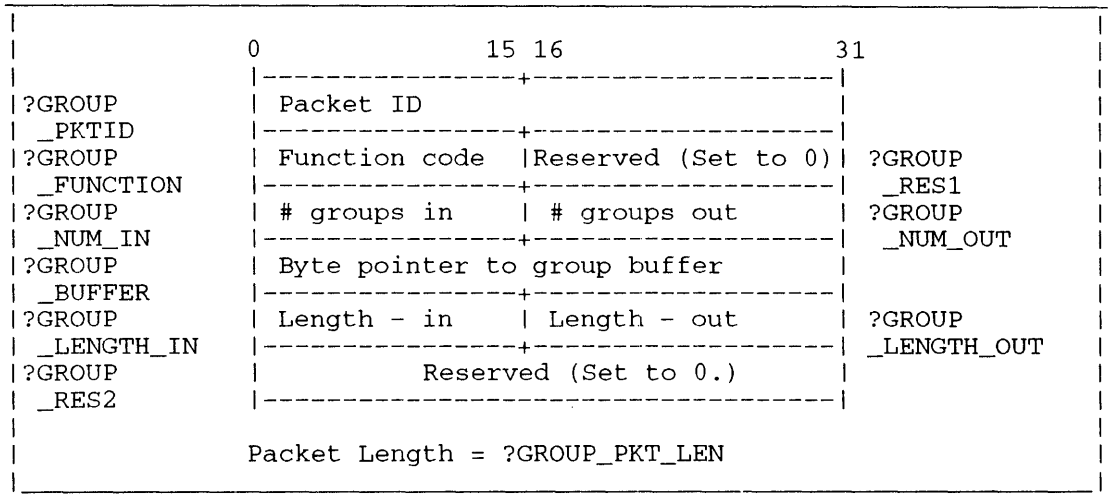


Figure 2–71. Structure of ?GROUP Packet

Logging

The operating system logs the replace function of the ?GROUP system call. The log entry consists of the standard header with the ?LGPRPL event code. Figure 2–72 shows the log entry format. Although the log entry subcode corresponds to the ?GROUP function code, it is not the same value because of the different numbering scheme used for the log entry subcode.

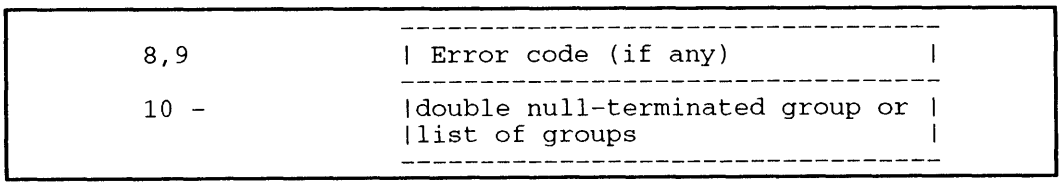


Figure 2–72. Structure of ?GROUP log entry

Table 2–50. Contents of ?GROUP Packet

Offset	Contents
?GROUP_PKTID	Contains the packet ID - ?GROUP_PKT_PKTID
?GROUP_FUNCTION	The values for the offset are ?GROUP_GET_LIST function returns in the group buffer the current group access control list of the calling process. ?GROUP_REPLACE function replaces the current group access control list with the list specified in the group buffer.
?GROUP_NUM_IN	Set to the number of groups in the group access control list to use as a replacement when using the ?GROUP_REPLACE function. A group list can contain up to ?GROUP_NUM_MAX number of names.
?GROUP_NUM_OUT	Returns the number of groups when using the ?GROUP_GET_LIST function.
?GROUP_BUFFER	Byte pointer to the group buffer address. ?GROUP_MIN_BUFFER_BYTE_LENGTH and ?GROUP_MAX_BUFFER_BYTE_LENGTH specify the size of the group buffer.
?GROUP_LENGTH_IN	Set to the number of bytes in the group buffer when using the ?GROUP_REPLACE function.
?GROUP_LENGTH_OUT	Returns the number of bytes in the group buffer when using the ?GROUP_GET_LIST function.

16–Bit System Call Support

The 16–bit version of ?GROUP supports RMA access. The 16–bit version uses the same packet offset as the 32–bit call. The 16–bit version also expects a ring field with the 16–bit addresses.

Notes

- See the description of ?PROC in this chapter, which initializes the group access control list for a process.
- See the description of ?XPSTAT in this chapter, which returns the group access control list of a process.
- PARU_LONG.SR defines the 32–bit offsets and parameters that appear in the packet. (PARU_LONG.H for users of C.)

?GSHPT — Lists the current shared partition size.

?GSHPT
error return
normal return

Input

None

Output

AC0 Page number of the first shared page in the calling process's current ring

AC1 Total number of shared pages in the calling process's current ring

AC2 Undefined

Error Codes in AC0

No error codes are currently defined.

Why Use It?

?GSHPT lets you determine the amount of logical shared memory the operating system has allocated to your process. Also, you can use ?GSHPT with ?SSHPT, which establishes a new shared partition in the caller's logical address space.

Who Can Use It?

There are no special process privileges needed to issue this call, and there are no restrictions concerning file access.

What It Does

?GSHPT returns the total number of shared pages in the caller's current ring, and the logical page number of the first shared page. ?GSHPT has no input requirements.

The operating system calculates the logical page numbers and returns them in AC0 and AC1. These page numbers range from 1 through 262,143 shared pages for Ring 7 and from 1 through 512 shared pages for Rings 4 through 6. (Page zero is always unshared.) If the caller's logical address space contains no shared pages, the operating system returns 0 to both AC0 and AC1.

Notes

- See the description of ?SSHPT in this chapter.

?GSID
error return
normal return

Input

AC0	Reserved (Set to 0.)
AC1	Reserved (Set to 0.)
AC2	Byte pointer to a 32–byte receive buffer for the system identifier

Output

AC0	Undefined
AC1	Undefined
AC2	Unchanged

Error Codes in AC0

ERMPR	System call parameter address error
ERVBP	Invalid byte pointer passed as a system call argument

Why Use It?

You can use ?GSID to get the system identifier for a particular system. This can be useful if you don't know what system you are on.

Who Can Use It?

There are no special process privileges needed to issue this call, and there are no restrictions concerning file access.

What It Does

?GSID returns the current system identifier to a receive buffer in your logical address space. The system identifier is an ASCII character string 32 bytes long. The string returned by ?GSID is exactly the same as the one set by ?SSID. By convention, anything appearing after the first null (if any) in the system identifier is ignored by most processes. (If the identifier is less than 32 characters long, the system puts a null terminator on the end; if the identifier is 32 characters long, no null is added.) The operator process (PID 2) can define the system identifier with the ?SSID system call.

Before you issue ?GSID, set up a 32–byte receive buffer for the identifier in your address space, and load AC2 with a byte pointer to the buffer.

Notes

- See the description of ?SSID in this chapter.

?GTACP
error return
normal return

Operating System Differences

AOS/RT32 always returns "+,OWARE" as the ACL.

Input

AC0	Byte pointer to the target file's pathname
AC1	One of the following: • Byte pointer to the target username • -1 to get the caller's access privileges
AC2	Reserved (Set to 0.)

Output

AC0	One or more of the following access privilege masks: ?FAOB Owner access ?FAWB Write access ?FAAB Append access ?FARB Read acess ?FAEB Execute access
AC1	Unchanged
AC2	Undefined

Error Codes in AC0

ERPRV	Caller not privileged for this action
ERVBP	Invalid byte pointer passed as a system call argument

Why Use It?

Because ?GTACP returns the ACL for a specific file and username, you can issue ?GTACP before you issue ?SACL, which sets the current ACL for a file, or before you issue ?DACL, which sets the default ACL.

Who Can Use It?

You need Execute access to a file's directory to determine your own access rights to a file. You need Superuser privilege to determine another user's access rights to a file.

What It Does

Under AOS/VS, ?GTACP returns bit masks in AC0 that indicate the target username's access privileges to the target file. Under AOS/RT32, ?GTACP returns "+,OWARE" as the ACL.

?GTAP Continued

Before you issue ?GTACP, specify the target file's pathname and the target username in your logical address space. Load AC0 with a byte pointer to the pathname, and load AC1 with a byte pointer to the username. The user parameter files, PARU.32.SR and PARU.16.SR, define the privilege bits as follows:

Mask	Bit Value	Meaning
?FACO	1B(?FAOB)	Owner access
?FACW	1B(?FAWB)	Write access
?FACA	1B(?FAAB)	Append access
?FACR	1B(?FARB)	Read access
?FACE	1B(?FAEB)	Execute access

Notes

- See PARU.32.SR or PARU.16.SR.

?GTIME — Gets the time, date, and time zone.

AOS/VS

?GTIME *[packet address]*
error return
normal return

Input

AC0	Reserved (Set to 0.)
AC1	Reserved (Set to 0.)
AC2	Address of the ?GTIME packet, unless you specify the address as an argument to ?GTIME

Output

AC0	Unchanged
AC1	Unchanged
AC2	Address of the ?GTIME packet

Error Codes in AC0

ERICD	Illegal function code
ERPVS	Packet version not supported (bad value in ?TIME_PKT.PKT_ID)
ERRVN	Reserved value not zero
ERVWP	Invalid word pointer passed as system call argument

Why Use It?

Use this system call to obtain information to postmark a message or to put a header on a listing page showing when your program created it.

Who Can Use It?

There are no special process privileges needed to issue this call, and there are no restrictions concerning file access.

What It Does

This system call returns the current time, date, and time zone. Time and date are based on prior operator input, and typically are the date and time at the site with the ECLIPSE MV/Family hardware on which the operating system is running.

Time zones are based on Universal Time (UTC). This is the international standard time derived from solar time at the meridian passing through Greenwich, England (the prime meridian, longitude 0 degrees). An older name for Universal Time is Greenwich Mean Time (GMT).

Time zones decrement as a person moves west of Greenwich, England; they increment as the person moves east of this city. For example, Mountain Standard Time is 7 hours west of Greenwich. Its time code is –7:00, so ?GTIME returns –7 and 0 respectively in offsets ?TIME_PKT.ZONE_HOUR and ?TIME_PKT.ZONE_MINUTE of the parameter packet. This information might be useful to someone in Phoenix, Arizona (United States of America). For another example, Calcutta, India is in a time zone +5-1/2 hours east of Greenwich. ?GTIME would return 5 and 30. respectively in offsets ?TIME_PKT.ZONE_HOUR and ?TIME_PKT.ZONE_MINUTE of the parameter packet.

The description of system call ?NTIME contains the maximum and minimum values returned in eight consecutive offsets of the parameter packet for ?GTIME (and for ?NTIME). These offsets are between ?TIME_PKT.YEAR and ?TIME_PKT.ZONE_MINUTE inclusive.

You should realize that *all* the information that ?GTIME returns can change from one issuance of ?GTIME to another. For example, the time zone can change because of going on daylight savings time in the spring, and the operating system time can change because someone made a mistake during initialization.

A single issuance of ?GTIME returns all the information needed for a message time stamp for both ARPANET and X.400 protocols. However, the protocols use text strings while ?GTIME returns binary numbers.

16–bit programs should not issue ?GTIME because the packet for ?GTIME does not appear in PARU.16.SR.

Figure 2–73 shows the structure of the ?GTIME parameter packet, and Table 2–51 describes its contents.

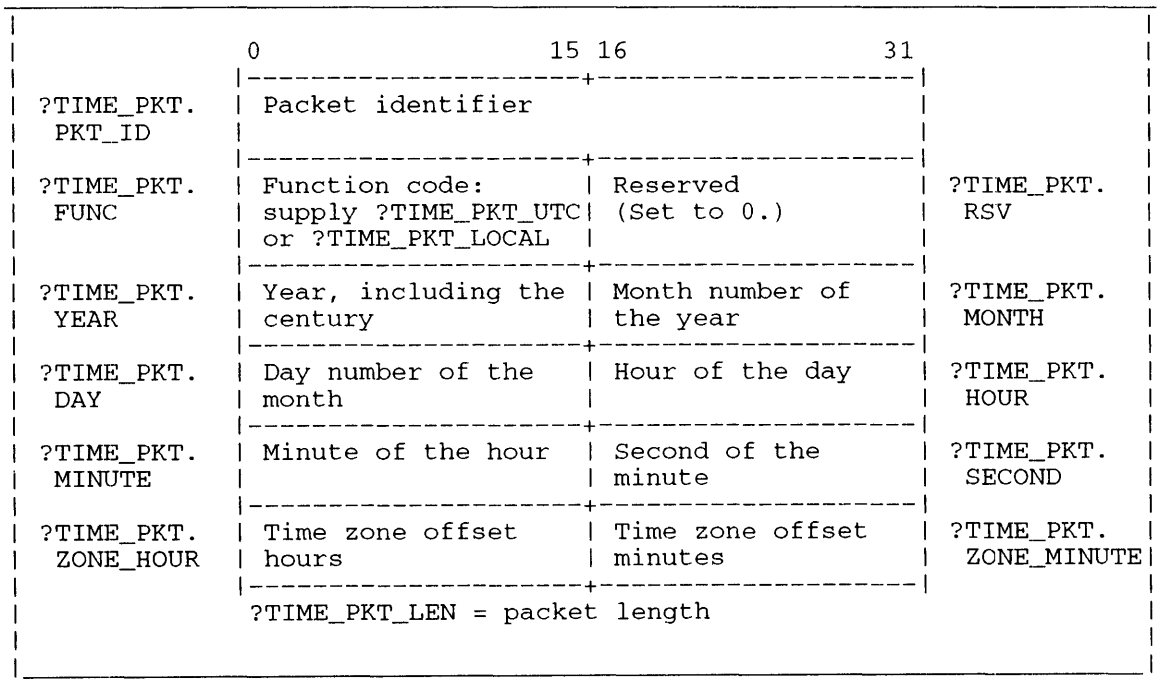


Figure 2–73. Structure of ?GTIME Packet

Table 2–51. Contents of ?GTIME Packet

Offset	Contents
?TIME_PKT.PKT_ID (doubleword)	Packet identifier. Place ?TIME_PKT_PKTID here.
?TIME_PKT.FUNC	Function code. To obtain Universal Time, place ?TIME_PKT_UTC here; to obtain local time, place ?TIME_PKT_LOCAL here.
?TIME_PKT.RSV	Reserved. (Set to 0.)
?TIME_PKT.YEAR	The operating system returns the current year as a number greater than 1967.
?TIME_PKT.MONTH	The operating system returns the current month as a number between 1 and 12 (1 for January, 2 for February, ..., 12 for December).
?TIME_PKT.DAY	The operating system returns the current day number of the month (1 for the 1st, 2 for the 2nd, ..., 31 for the 31st).
?TIME_PKT.HOUR	The operating system returns the current hour number of the day as a number between 0 and 23, inclusive (i.e., a 24-hour clock).
?TIME_PKT.MINUTE	The operating system returns the current minute number of the hour as a number between 0 and 59, inclusive.
?TIME_PKT.SECOND	The operating system returns the current second number of the minute as a number between 0 and 59, inclusive.
?TIME_PKT. ZONE_HOUR	The operating system returns the current offset in hours relative to Universal Time. Western Hemisphere zones have negative values, Eastern Hemisphere zones have positive values, and the zone whose center is the prime meridian has a zero value. For example, West Germany is one time zone east of the prime meridian; the operating system will return 1 in this word for the 1 hour offset (i.e., difference).
?TIME_PKT. ZONE_MINUTE	The operating system returns the current offset in minutes relative to ?TIME_PKT.ZONE_HOUR. For most countries this value is zero, but for a few -- such as India -- this value is nonzero. India is five time zones east of the prime meridian. When it is 0300 (3:00 am) in Greenwich it is 0830 (8:30 am) in India; consequently ?TIME_PKT.ZONE_MINUTE contains 30 (decimal) and ?TIME_PKT.ZONE_HOUR contains 5.

Notes

- See the descriptions of ?NTIME and ?RTODC in this chapter.

?GTMES *[packet address]*
error return
normal return

Input

AC0	Reserved (Set to 0.)
AC1	Reserved (Set to 0.)
AC2	Address of the ?GTMES packet, unless you specify the address as an argument to ?GTMES

Output

AC0	Dependent on packet's input values (See Table 2–52.)
AC1	Dependent on packet's input values (See Table 2–52.)
AC2	Address of the ?GTMES packet

Error Codes in AC0

ERVBP	Invalid byte pointer passed as a system call argument
ERVWP	Invalid word pointer passed as a system call argument

Why Use It?

?GTMES allows you to access the initial IPC message that the CLI sends when it creates a new process. This message is an edited version of the CLI command used to create the process. Depending on your input specifications, you use ?GTMES to get a specific argument in the CLI command line and to determine which switches, if any, modify it. ?GTMES also returns the message that another father process sends when it creates a son with the ?PROC system call.

Who Can Use It?

There are no special process privileges needed to issue this call, and there are no restrictions concerning file access.

What It Does

When you create a process by executing a CLI command, the CLI sends the new process an initial message that contains an edited version of the CLI command. ?GTMES gives a CLI son process access to that message or, depending on your input specifications, access to

- The length of the message, in words.
- The number of arguments in the CLI command (argument count) and any simple or keyword switches used to modify a particular argument. A keyword switch is a two–part switch of the form

  /keyword = value.

  For example, /L=filename is a keyword switch.
- One of the CLI command line arguments, minus the switches.

Figure 2–74 shows the structure of the ?GTMES packet, and Table 2–52 describes each offset and parameter in the packet.

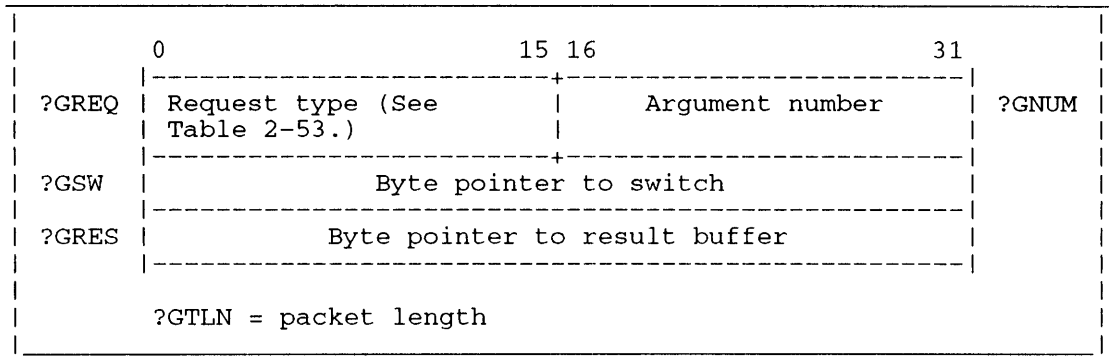


Figure 2–74. Structure of ?GTMES Packet

Most of the ?GTMES parameters apply to CLI messages. However, offset ?PIPC in the ?PROC packet passes an IPC header from the ?PROC caller to the son, which allows you to use ?GTMES to obtain an IPC message sent by father process (other than the CLI) when it creates a son.

Offset ?GREQ describes the ?GTMES request; that is, the kind of information you want the operating system to return. Choose only one request type for offset ?GREQ. Table 2–53 lists the possible request types and their meanings.

Table 2–52. Contents of ?GTMES Packet*

Offset	Contents
?GREQ	Request type: specifies the information you want the OS to return. (See Table 2-53 for request types.)
?GNUM	Argument in the CLI command line; specify the argument you want by its relative position in the command line. For example, ?GNUM = 0 specifies the first argument; that is, program name.
?GSW (doubleword)	Byte pointer to a simple or keyword switch you want to test for. (Use ?GSW only if request type is ?GTSW; see Table 2-53.) A keyword switch can be no longer than 32 bytes, including the terminating null. DEFAULT = 0 (do not test for a switch).
?GRES (doubleword)	Byte pointer to buffer you want to receive the results. (The OS terminates the results string with a null.) DEFAULT = -1 (pass results to AC0 and AC1 only) (See Table 2-54.)

* There is no default unless otherwise specified.

Table 2–53. Input Parameters for Offset ?GREQ (Request Types)

Request	Meaning
?GMES	Copy the entire message to the message buffer specified in offset ?GRES (unless this offset contains -1). Return the message flag, if present, to AC0, and the length of the message (number of words) to AC1. The message flag is ?GFCF Message is in CLI format (the first argument is program name).
?GCMD	Read the edited CLI command line into the buffer specified in offset ?GRES (unless these offsets contain -1). Return the byte length of the edited command line to AC1. Some CLI commands, such as EXECUTE, are special cases. They appear near the end of this explanation of ?GTMES.
?GCNT	Get the argument count (number of arguments in the CLI command line, excluding the program name) and return this value to AC0.
?GARG	Copy the command line argument specified in ?GNUM, minus switches, to the ?GRES buffer area (if that area exists). If the argument consists entirely of decimal digits, the OS converts it to binary, returns the results to AC1, and returns the argument's byte length in AC0. (The largest possible value is 2,147,483,646. A value of 2,147,483,647 has the OS return error code ERISV to AC0 and the number of characters to AC1. Values greater than 2,147,483,647 begin from 0 -- for example, 2,147,483,648 has the same effect as 0 and 2,147,483,649 has the same effect as 1.)
?GTSW	Test the simple or keyword switch referred to by ?GSW to see whether it modifies the argument cited in ?GNUM. Return the following test results to AC0: -1 switch not found. 0 switch found. > 0 byte length of the keyword switch. If the OS finds a keyword switch, it returns that switch's value, terminated by a null, to the ?GRES buffer.

(continued)

Table 2–53. Input Parameters for Offset ?GREQ (Request Types)

Request	Meaning
?GTSW (cont.)	If the switch value consists entirely of decimal digits, the OS converts it to binary and returns the result to AC1. (The largest possible value is 65,535. If the keyword switch is larger than 65,535, the OS not only returns an error, but it also returns the number of characters in the keyword switch to AC1.) If the switch appears in the argument more than once, the OS returns information for the first occurrence only.
?GSWS	Examine the ?GNUM argument for simple switches or alphabetic keyword switches. Set corresponding low-order bits in AC0 and AC1 for each switch. For example: In AC0 1S16 means /A or /a 1S17 means /B or /b ... 1S31 means /P or /p In AC1 1S16 means /Q or /q 1S17 means /R or /r ... 1S25 means /Z or /z Note that both lowercase and uppercase simple switches will set the bits. (Bits 26 through 31 in AC1 are always 0.)
?GDLC	Disable lower- to uppercase conversion for requests ?GCMD, ?GCNT, ?GARG, ?GTSW, and ?GSWS. Specify ?GDLC along with these requests to provide the disabling. You MUST select ?GDLC on the first call and on all subsequent calls for which you need lowercase distinction. As soon as you issue one ?GTMES call without selecting ?GDLC, the operating system converts the CLI message to uppercase letters for this call and for all subsequent calls, regardless of whether you have selected ?GDLC.

(concluded)

As Table 2–53 shows, several of the ?GTMES requests use the buffer area specified by offset ?GRES. If you choose the default buffer specification (by setting ?GRES to –1), the OS returns information only to AC0 and AC1. Table 2–54 summarizes the information returned to the buffer area and to AC0 and AC1.

Table 2–54. Output from ?GTMES Requests

Request Type	AC0	AC1	?GRES Buffer
?GMES	Message Flag (if message is in CLI format).	Total word length of message.	Entire message.
?GCMD	Unchanged.	Byte length of CLI command line.	CLI command line
?GCNT	Number of arguments (excluding argument 0 and program name).	Unchanged.	N/A.
?GARG	Byte length of argument specified in ?GNUM (excluding terminating null).	Binary equivalent of argument, if argument is decimal value not greater than 65,535. If argument value is greater than 65,535, return number of characters in argument. If argument is non-numeric, -1.	Actual argument string.
?GTSW	Switch test results -1 No switch. 0 Simple switch. > 0 Byte length of keyword switch.	Binary equivalent of keyword switch, if keyword switch is decimal value not greater than 65,535. If keyword switch is greater than 65,535, return number of characters in keyword switch. If keyword switch is nonnumeric, -1.	Actual keyword switch.
?GSWS	Switch value 1S16 = /A 1S17 = /B ... 1S31 = /P	Switch value 1S16 = /Q 1S17 = /R ... 1S25 = /Z (Bits 26-31 = 0)	N/A.
?GDLC	Output depends on what function you previously specified along with ?GDLC. For example, if you specified ?GCMD with ?GDLC, the output would follow the output for that command (AC0 is unchanged, AC1 contains the byte length of the CLI command line, and buffer ?GRES contains the CLI command line in upper- and lowercase).		

Request type ?GMES copies the entire message to the ?GRES area (if it exists) and copies the message length to AC1. If the message is in CLI format, this request type also passes the ?GFCF flag to AC0. Thus, request type ?GMES lets you determine the format (CLI or not) and length of the message.

If you specify a buffer area and select a request type other than ?GMES, the CLI returns an edited version of the message with the following format:

- One comma separating each argument.
- The message word aligned and terminated by a null (or two nulls, if necessary to ensure an even number of bytes).
- The high–order bit of every byte set to 0 (not used for parity).
- Lowercase characters converted to uppercase characters (unless you select request type ?GDLC).
- Without the following characters: space, <, >, [,], (,), ;, New Line, carriage return, form feed, embedded nulls.

If you select request type ?GCMD and the CLI command was EXECUTE, XEQ DEBUG, PROCESS, or CHAIN, the operating system removes the command name from the edited message.

Request type ?GTSW tests to see if a simple or keyword switch modifies the argument in offset ?GNUM. Be sure to set ?GNUM to an argument number (that is, to the relative position of the argument) within the range 0 through n, where 0 is the first argument (program name, for CLI command), and n is the last argument. Use offset ?GSW as a byte pointer to the switch you wish to test. When you specify the switch, remove the initial backslash and terminate the switch with a null. For example, write /L as L<0>.

Request type ?GSWS also tests for the presence of a switch in the argument and, in addition, sets a corresponding bit in the low–order portion of either AC0 or AC1.

To find out whether or not the message is written in CLI format, select ?GMES as the request type and set ?GRES to –1 (no buffer area). Then examine the output value of AC0. If AC0 contains the message flag ?GCFC, the message is in CLI format.

See sample programs BOOMER, DLIST, and TIMEOUT in Appendix A. They all issue system call ?GTMES.

Sample Packet

The following sample packet gets a filename argument.

```
PKT:    .BLK    ?GTLN           ;Allocate enough space for the packet
                                ;(packet length = ?GTLN).
        .LOC    PKT+?GREQ       ;Request type. (See Table 2-53.)
        .WORD   ?GARG           ;Get the command line argument
                                ;specified in ?GNUM.
        .LOC    PKT+?GNUM       ;Argument number.
        .WORD   -1              ;Specify argument number at ?GTMES
                                ;time.
        .LOC    PKT+?GSW        ;Byte pointer to switch you want to
                                ;test for.
        .DWORD  0               ;Not used. (You only use ?GSW when
                                ;the request type is ?GTSW.)
        .LOC    ?GRES           ;Byte pointer to buffer that receives
        .DWORD  -1              ;the results. Specify at ?GTMES time.
        .LOC    PKT+?GTLN       ;End of packet.
```

Notes

- See the description of ?PROC in this chapter.

?GTNAM — Returns symbol closest in value to specified input value.

AOS/VS

?GTNAM
error return
normal return

Input

AC0	Input value
AC1	Contains the following: • Bits 16 through 23 contain the byte length of the receive buffer (specified in AC2) • Bits 24 through 31 contain the channel number of the .ST file
AC2	Byte pointer to a receive buffer for the symbol name

Output

AC0	Offset between the input value that you specify and the symbol in the receive buffer
AC1	Unchanged
AC2	Unchanged

Error Codes in AC0

ERICN	Invalid channel number
ERIRB	Insufficient room in buffer (The receive buffer is too small.)
ERVBP	Invalid byte pointer passed as a system call argument

Why Use It?

?GTNAM allows you to search the symbol table (.ST file) for a symbol that approximates a specific value, without writing a routine for this purpose. For example, you can use ?GTNAM in a utility program that references symbols in the .ST file. ?GTNAM lets you reference the .ST file without knowing its format.

Who Can Use It?

There are no special process privileges needed to issue this call, and there are no restrictions concerning file access.

What It Does

?GTNAM searches the symbol table (.ST file) and returns the symbol closest in value to the value you placed in AC0.

The OS returns the symbol name to the receive buffer that you specify in AC2, and also returns the offset between the input value and the symbol value to AC0.

Before you issue ?GTNAM, perform the following steps:

1. ?SOPEN your program's .ST file.
2. Set up a receive buffer for the symbol name and load its address into AC2.
3. Specify the input value in AC0.
4. Specify the length of the receive buffer and the channel number of the .ST file in AC1. (The ?SOPEN system call returns the channel number.)

The symbol that ?GTNAM returns can consist of up to 32 bytes. Thus, you should define a receive buffer of at least 32 bytes. If the operating system cannot find a symbol that is less than the input value, it returns a 32–byte null string to the receive buffer.

Notes

- See the descriptions of ?SOPEN and ?GTSVL in this chapter.

?GTOD — Gets the time of day.

?GTOD
error return
normal return

Input

None

Output

AC0	Seconds from 0 through 59
AC1	Minutes from 0 through 59
AC2	Hour from 0 through 23

Error Codes in AC0

No error codes are currently defined.

Why Use It?

You can use ?GTOD to get the current time. Next, you can use it for output on file listings.

Who Can Use It?

There are no special process privileges needed to issue this call, and there are no restrictions concerning file access.

What It Does

?GTOD gets the current time from the system clock and returns it to the accumulators in standard notation. The value for the hour can range from 0 (midnight) through 23 (11 p.m.), and the values for seconds and minutes can range from 0 through 59.

?GTRUNCATE *[packet address]*
error return
normal return

Input

AC0	Reserved (Set to 0.)
AC1	Channel number of the disk file
AC2	Address of the ?GTRUNCATE packet, unless you specify the address as an argument to ?GTRUNCATE.

Output

AC0	Undefined
AC1	Unchanged
AC2	Address of the ?GTRUNCATE packet

Error Codes in AC0

EREOF	End of file (You tried to set the EOF (end–of–file) past the current EOF.)
ERFNO	Channel not open
ERMUS	Multiple users of file; cannot truncate
ERSHR	Shared file; cannot truncate
ERSIM	Simultaneous requests on same channel
ERVWP	Invalid word pointer passed as a system call argument
ER_FS_DIRECTORY_NOT_AVAILABLE	Directory not available because the LDU was force released (AOS/VS II only)
ER_FS_TLA_MODIFY_VIOLATION	Attempt to modify an AOS/VS II file with ?ODTL value supplied in ?GOPEN packet

Why Use It?

You can use ?GTRUNCATE to reduce the size of a disk file that is currently open for block I/O. Thus, ?GTRUNCATE allows you to reclaim disk space that you no longer need.

Who Can Use It?

There are no special process privileges needed to issue this call. You must have access to the file's channel number (via ?OPEN or ?GOPEN) and you must have Write access to the file.

What It Does

?GTRUNCATE truncates (shortens) disk files. The caller passes a channel number and a pointer to a new ?GTRUNCATE packet. The packet contains the new size of the file in bytes. This new size must be less than or equal to the current size of the file or the operating system returns error code EREOF (end of file).

If you need to truncate a magnetic tape file, use ?TRUNCATE.

The following restrictions apply to files that you want to truncate:

- The file must have a use count of +1.

?GTRUNCATE Continued

- The file must not be open for shared I/O.
- The file should have been opened with ?GOPEN instead of ?OPEN.

Figure 2–75 shows the structure of the ?GTRUNCATE packet, and Table 2–55 describes its contents.

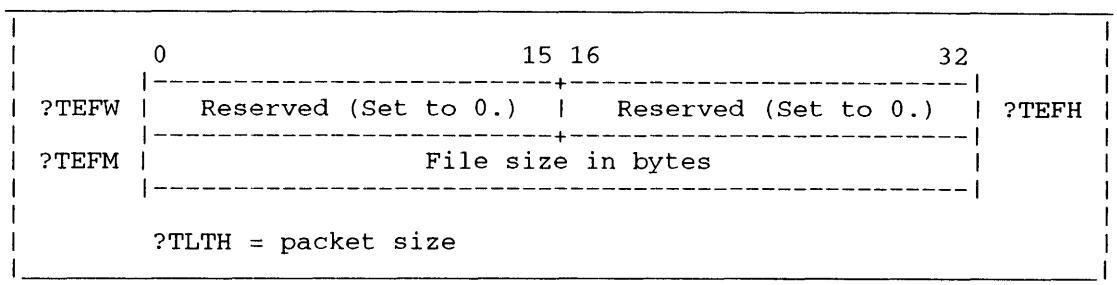


Figure 2–75. Structure of ?GTRUNCATE Packet

Table 2–55. Contents of ?GTRUNCATE Packet*

Offset	Contents
?TEFW	Reserved. (Set to 0.)
?TEFH	Reserved. (Set to 0.)
?TEFM (doubleword)	New file size in bytes. The new file size must be less than or equal to the current (old) file size.

* There is no default unless otherwise specified.

When you use ?GTRUNCATE, be sure to reset the values of offset ?PRNH and ?PRCL in the block I/O packet.

Notes

- See the descriptions of ?OPEN and ?GOPEN in this chapter.
- See the descriptions of ?PRDB/?PWRB and ?BLKIO in this chapter for information on the block I/O packet.

AOS/VS

?GTSVL
error return
normal return

Input

AC0	Reserved (Set to 0.)
AC1	Contains the following: • Bits 16 through 23 contain the byte length of the symbol name (specified in AC2) • Bits 24 through 31 contain the channel number of the .ST file
AC2	Byte pointer to the name of the target symbol

Output

AC0	Value of the target symbol
AC1	Unchanged
AC2	Unchanged

Error Codes in AC0

ERICN	Invalid channel number
ERSNF	Symbol not found in ?GTSVL system call (The symbol is not in the .ST file.)
ERVBP	Invalid byte pointer passed as a system call argument

Why Use It?

?GTSVL lets you retrieve the value of a particular user symbol, as recorded in your program's .ST file. Like ?GTNAM, ?GTSVL allows you to reference the .ST file without knowing its exact format.

Who Can Use It?

There are no special process privileges needed to issue this call, and there are no restrictions concerning file access.

What It Does

?GTSVL returns the symbol table value of a particular user–defined symbol. Before you issue ?GTSVL, perform the following steps:

1. ?SOPEN your program's .ST file (symbol table).
2. Specify the target symbol name in your logical address space and load AC2 with a byte pointer to the symbol name.

3. Specify the length of the symbol name and the channel number for the .ST file in AC1. (?SOPEN returns the channel number.)

The operating system returns the value of the specified symbol in AC0.

You must issue ?SCLOSE against the file with bit 0 of AC1 set. ?GTSVL issues ?SPAGE on the symbol table file, so you need to release the pages.

Notes

- See the descriptions of ?SOPEN, ?SCLOSE, and ?GTNAM in this chapter.

?GUHPI Gets unique hardware processor identification.

AOS/VS

?GUHPI *[packet address]*
error return
normal return

Input

AC0	Reserved (Set to 0.)
AC1	Reserved (Set to 0.)
AC2	Address of the ?GUHPI packet, unless you specify the address as an argument to ?GUHPI

Output

AC0	Unchanged
AC1	Unchanged
AC2	Address of the ?GUHPI packet

Error Codes in AC0

ERICM	Illegal system command (LAN ID does not exist or OS cannot read it)
ERIRB	Insufficient room in buffer (less than 6 bytes)
ERMPR	Invalid parameter address (bad packet pointer, etc.)
ERPKT	Illegal packet ID

Why Use It?

Proprietary software can use this call to identify a particular computer installation. For example, suppose that you program for a software vendor and want to ensure that your software runs only on computers that are licensed for it. Your program can issue this call and compare the results to an internal list of unique ID numbers of licensed computers.

Who Can Use It?

There are no special process privileges needed to issue this call, and there are no restrictions concerning file access.

What It Does

The ?GUHPI system call allows your process to identify uniquely the hardware processor that it is running on. The ID number the call returns is based on a Local Area Network (LAN) ID.

This call succeeds only on systems that have a LAN ID and a LAN controller. In other words, the system must be part of a LAN or else the call returns error code ERICM. Also, the call succeeds only on ECLIPSE MV/2000 DC and DS/7000–series computers.

Figure 2–76 shows the structure of the ?GUHPI parameter packet, and Table 2–56 describes its contents.

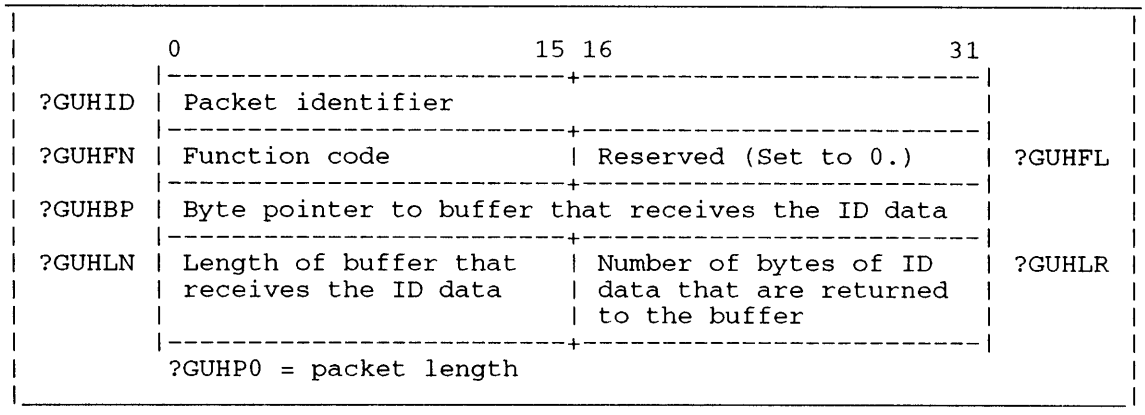


Figure 2–76. Structure of ?GUHPI Packet

Table 2–56. Contents of ?GUHPI Packet

Offset	Contents
?GUHID (double-word)	Packet identifier. Place ?GUID here.
?GUHFN	Function code. Not used (Set to 0.)
?GUHFL	Flags word. Reserved (Set to 0.)
?GUHBP (double-word)	Supply a byte pointer to the buffer that receives the ID data.
?GUHLN	Supply the number of bytes in the buffer that receives the ID data. This number must be at least 6. If it is more than 6, the operating system places the ID data left justified into the buffer.
?GUHLR	The operating system returns the number of bytes of ID data it has placed into the buffer that offset ?GUHBP points to.

?GUNM
error return
normal return

Input

AC0	One of the following: • –1 to obtain the username of the calling process • PID of the target process • Byte pointer to the name of the target process
AC1	One of the following: • –1 if AC0 contains a byte pointer • 0 if AC0 contains a PID • Any other value if AC0 contains –1
AC2	Byte pointer to the area that will receive the username (This area must be ?MXUN bytes long.)

Output

AC0	One of the following: • 0 if the target process is not in Superuser mode • 1 if the target process is in Superuser mode
AC1	Privilege word of the target process
AC2	Unchanged

Error Codes in AC0

ERMPR	System call parameter address error
ERPNM	Illegal process name format
ERPRH	Attempt to access a process not in hierarchy
ERVBP	Invalid byte pointer to name of target process passed in AC0

Why Use It?

?GUNM is useful if you know the simple process name or PID of a process, and you want to know the username associated with that process. (A full process name consists of the process's username and its simple process name.)

Who Can Use It?

There are no special process privileges needed to issue this call, and there are no restrictions concerning file access.

What It Does

?GUNM returns the target process's username terminated by a null to the location that you specify in AC2. Before you issue ?GUNM, load AC0 either with the PID of the target process or with –1 to obtain the caller's username. If you specify –1 in AC0, the operating system ignores AC1 because there is no need for a PID or byte pointer to the target's process name.

AOS/VS

?GVPID
error return
normal return

Input

AC0	PID
AC1	Host ID in Bits 17 through 31
AC2	Reserved (Set to 0.)

Output

AC0	Virtual PID
AC1	Unchanged
AC2	Unchanged

Error Codes in AC0

ERHNE	Host does not exist (You supplied an invalid host ID in AC1.)
ERPRE	Illegal system call parameter (You supplied an invalid PID in AC0.)

Why Use It?

?GVPID encodes a host–ID/PID pair into a virtual PID for use in subsequent system calls.

Who Can Use It?

There are no special process privileges needed to issue this call, and there are no restrictions concerning file access.

What It Does

?GVPID returns a virtual PID. A virtual PID is a 16–bit value that references the PID (supplied in AC0) on the remote system (identified by the host ID in AC1). If the operating system has already assigned a virtual PID to the process, then ?GVPID returns the old virtual PID.

?HNAME
error return
normal return

Case 1 — Given a host ID, return the hostname
Case 2 — Given a hostname, return the host ID
Case 3 — Return the local host ID and, optionally, the local hostname

Input

AC0 One of the following:

- Byte pointer to the buffer that you specify is to receive the hostname (The buffer must be at least ?MXHN bytes long.) (Cases 1 and 3.)
- Byte pointer to the buffer that contains the hostname (Case 2)
- 0 to return no local hostname (Case 3)

AC1 One of the following:

- Host ID in Bits 17 through 31 (Case 1)
- 0 (Case 2)
- –1 (Case 3)

AC2 Reserved (Set to 0.)

Output

AC0 Unchanged. (Case 1 —?HNAME returns the hostname to the buffer that AC0 points to)

AC1 One of the following:

- Unchanged (Case 1)
- Host ID in Bits 17 through 31 (Case 2)
- Local host ID in Bits 17 through 31 (Case 3)

AC2 One of the following:

- Length (excluding the null terminator) of the hostname that ?HNAME returned (Case 1)
- Undefined (Case 2)
- Unchanged (if input AC0 contains a 0) (Case 3)
- Length (excluding null terminator of local hostname) (Case 3)

?HNAME Continued

Error Codes in AC0

ERHID	Illegal host specification
ERHNE	Host does not exist
ERIHN	Illegal hostname

Why Use It?

?HNAME allows you to determine host information by specifying the host ID or the hostname.

Who Can Use It?

There are no special process privileges needed to issue this call, and there are no restrictions concerning file access.

What It Does

?HNAME lets you get a hostname if you have the host ID or vice versa.

When you specify a host ID in Bits 17 through 31 of AC1, ?HNAME returns the hostname to the buffer pointed to by AC0. When you specify a byte pointer to the hostname in AC0, ?HNAME returns the host ID in Bits 17 through 31 of AC1.

However, ?HNAME also lets you get the local host ID and, optionally, the local hostname. To do this, specify a byte pointer to the buffer that is to receive the local hostname in AC0 (if you want the operating system to return the local hostname) or 0 in AC0 (if you do not want the operating system to return the local hostname) and –1 in AC1. Then, the operating system returns the local host ID in Bits 17 through 31 of AC1 and AC2 contains either a 0 (if AC0 contained 0 on input) or the length of the local hostname, excluding the null terminator.

?IDEF
error return
normal return

Input

AC0 Contains the following:

- Bit 1 is a flag bit:

  Bit 1 = 0 if AC2 specifies the number of data channel map slots needed (no map definition table)

  Bit 1 = 1 if AC2 points to a data channel map definition table

- Bit 2 is a flag bit for 16 bit processes:

  Bit 2 = 1 if you want to use the extended packet

- Bits 3 through 31 contain the device code for the user–defined device in the range from 1 through 191 (= 277 octal)

AC1 Bits 1 through 31 contain the address of the device's DCT; set Bit 0 to 1 if the device is a data channel (DCH) or burst multiplexor channel (BMC) device

AC2 One of the following:

- Address of the map definition table
- Number of map slots needed (no map definition table) (each map slot accesses 1K words)

Output

AC0 Unchanged

AC1 Unchanged

AC2 Unchanged

If your program executes as a 16–bit process, then the bit numbers in AC0 and AC1 change from above as follows:

Bit 0	would read Bit 16
Bit 1	would read Bit 17
Bits 1 through 31	would read Bits 17 through 31
Bits 2 through 31	would read Bits 18 through 31

Error Codes in AC0

ERDCH	Data channel map full
ERDNM	Illegal device code (The device code is outside the legal range (1 through 191).)
ERIBS	Device already in use
ERPRE	Invalid system call parameter
ERPRV	Caller is not privileged for this action
ERPTY	Illegal process type

Why Use It?

?IDEF lets you establish an interface between the operating system and a device it does not support. ?IDEF and the other user–device system calls are particularly useful if you have applications–specific peripheral devices for which you have written special device–driver routines.

Who Can Use It?

The caller must be a resident process and must have privilege ?PVDV to use ?IDEF. There are no restrictions concerning file access.

What It Does

?IDEF defines a user device and its device control table (DCT). The operating system builds an internal DCT based on your DCT specifications, and enters this into its interrupt vector table.

Before you issue ?IDEF, set up a device control table in your logical address space and load its address into Bits 1 through 31 of AC1. (See Figure 2–77 for the DCT format if you want to issue ?IDEF from a 32–bit process. See Figure 2–78 if you want to issue ?IDEF from a 16–bit process.) Be sure to set offset ?UDVIS in the DCT to the address of the interrupt service routine for the new device, and define the interrupt service mask in offset ?UDVMS. In addition, you must set offset ?UDVBX (the mailbox) to 0.

For devices that reside on the secondary IOCs (i.e., device codes 64.–191.), you must use PIO instructions to communicate with your device. NOVA I/O is limited to the first IOC.

You may extend the DCT to include a word pointer to a device termination routine. In this case control passes to the routine if your process traps or terminates. This transfer of control prevents a runaway ?IDEF–specified device from altering system databases. Also, the environment would be the same at process termination time as if the device had just requested an interrupt. You might specify the extended DCT if you are writing an interrupt service routine (ISR) for a device such as an intelligent asynchronous controller (IAC) or an intelligent synchronous controller (ISC).

To extend the DCT, place ?UDLX in offset ?UDRS and place a word pointer to your device termination routine in offset ?UDDTR. The length of the extended packet is ?UDLE (= ?UDLN+?UDLX) words.

To specify the standard DCT, place 0 in offset ?UDRS and allow ?UDLN words for the unextended packet. In other words, programs written before you could extend the DCT (effective with AOS/VS Revision 7.00) will not have to change.

If you issue ?IDEF against a device that AOS/VS is using, you receive error code ERIBS. AOS/VS uses devices that were identified to it during the system generation process. An example is an MTD magnetic tape controller. AOS/VS also uses the following devices.

Device Mnemonic	Code	Description
BMC	–	Burst Multiplexor Channel
CPU	77	Central Processor
DCH	–	Data Channel
PIT43	Programmable Interval Timer	
RTC	14	Real–Time Clock
SCP	45	System Control Processor
TTI10	Primary Asynchronous Line Input	
TTO	11	Primary Asynchronous Line Output
UPSC	4	Universal Power Supply Controller

In addition, AOS/VS uses a DRT (Dual Receive/Transmitter, device code 34) on MV/1400 DC and MV/2000 DC computers. Another name for this device is DUART.

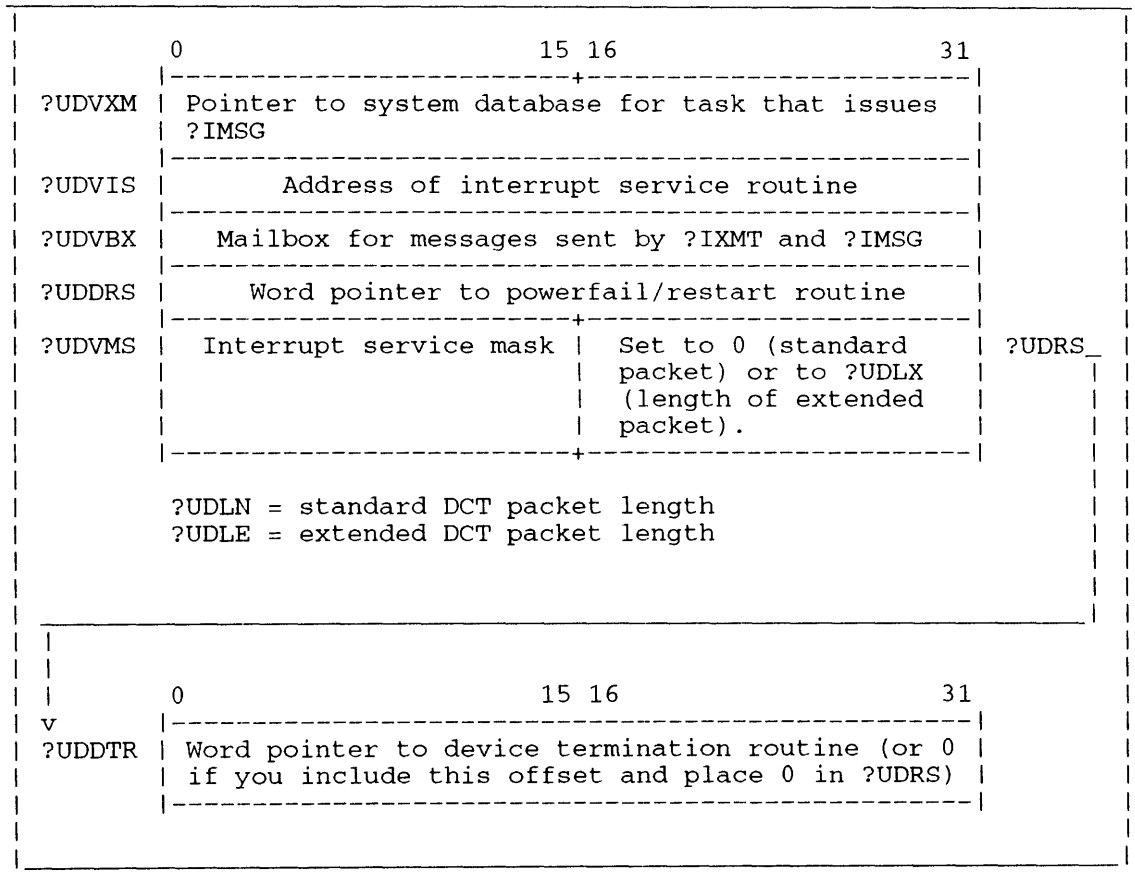


Figure 2–77. Structure of Device Control Table (DCT) for 32–Bit Processes

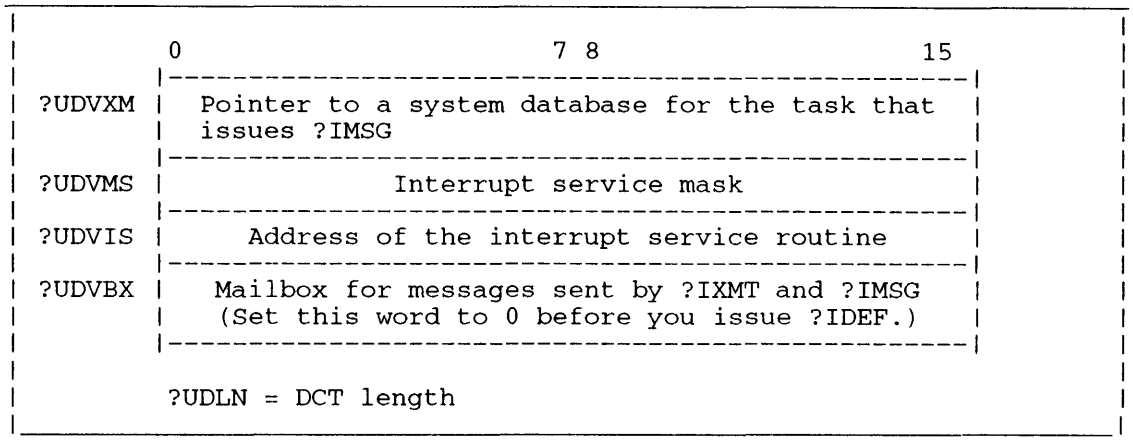


Figure 2–78. Structure of Device Control Table (DCT) for 16–Bit Processes

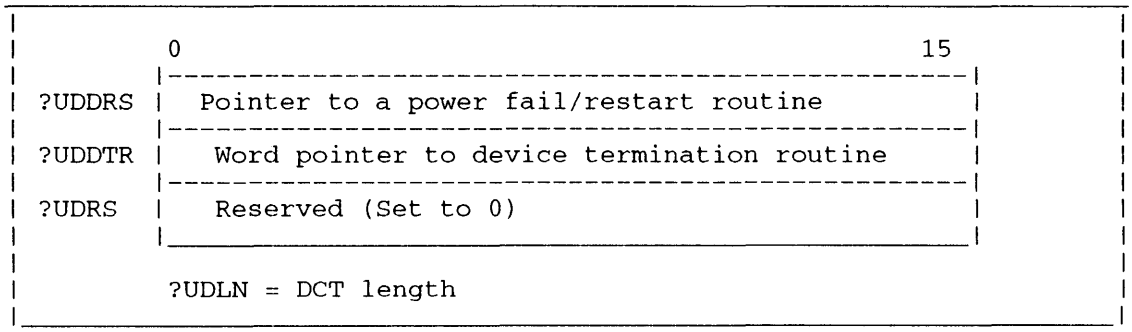


Figure 2–78.1. Structure of Extended Packet for 16–Bit Processes

Options

Set Bit 0 of AC1 if you want to use either the data channel (DCH) or the burst multiplexor channel (BMC) for I/O transfers to and from the new device. If you choose the DCH or BMC option, you must also define the number of map slots the device will need. You can load the map slot value into AC2 before you issue ?IDEF or you can set up a map definition table in your address space. If you set up a map definition table, load its address into AC2 before you issue ?IDEF. (If you use AC2, the operating system will allocate map slots in DCH map A.)

Set Bit 2 of AC0, if you want to use the extended packet to clear LBUS interrupts. Figure 2–78.1 shows the structure of the extended packet.

The map definition table specifies the first acceptable map slot for BMC or DCH transfers, and optionally, selects a particular DCH map (maps A through P). The map definition table can contain as many as eight entries. Each entry is ?UDELTH words long. The entire table (with eight entries) is ?UDLTH words long. (See Figure 2–79 for the structure of a map definition table entry and see Table 2–57 for a description of its contents.)

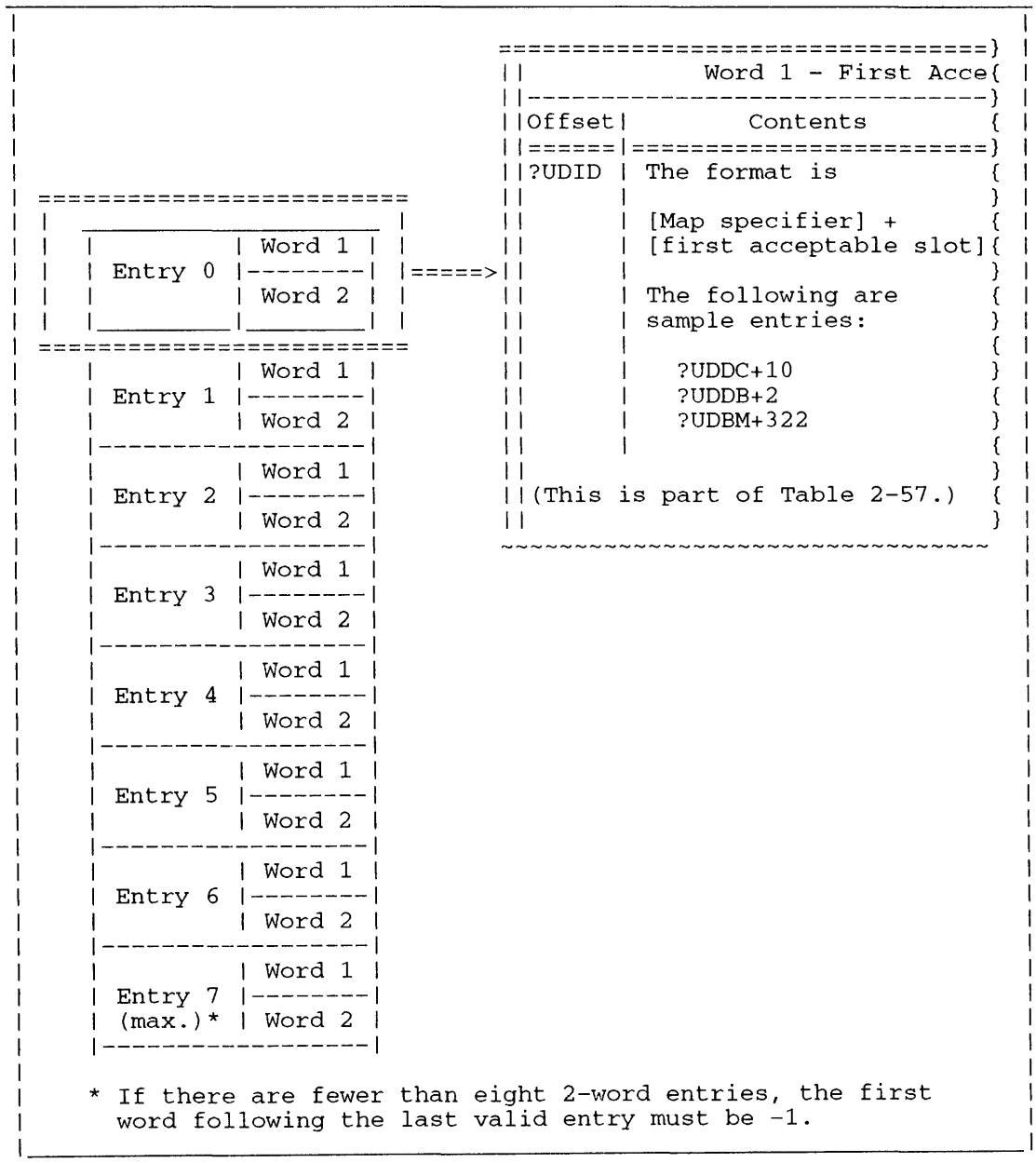


Figure 2-79. Structure of Map Definition Table

Table 2–57. Contents of Map Definition Table Entry

Word 1 - First Acceptable Map Slot		
Offset	Contents	Comments
?UDID	The format is [Map specifier] + [first acceptable slot] The following are sample entries: ?UDDC+10 ?UDDB+2 ?UDBM+322	Map specifier must be one of the following: - ?UDBM, which selects the BMC map. - ?UDDA, which selects the DCH A map. - ?UDDB, which selects the DCH B map. - ?UDDC, which selects the DCH C map. - ?UDDD, which selects the DCH D map. - ... - ?UDDP, which selects the DCH P map. First acceptable slot must be - From 0 through 1777 (octal), if your map specifier is ?UDBM. - From 0 through 37 (octal), if your map specifier is ?UDDA, ?UDDB, ?UDDC, or ?UDDD. The OS allocates the first contiguous group of slots on the map, starting with the first acceptable slot on the map that you selected. Then, the OS returns to you the first slot that it allocated in ?UDID.
Word 2 - Number of Map Slots Requested		
Offset	Contents	Comments
?UDNS	Number of map slots requested in range from 1 through 40 (octal).	The sum of the first acceptable slot plus the number of slots cannot be larger than the size of the map that you requested; that is, 40 (octal) for DCH or 2000 (octal) for BMC.

NOTE: If the OS cannot allocate all entries, then it does not allocate any entries.

Offset ?UDID, the map identifier word, defines which channel (the BMC or the DCH) the operating system should use for data transfers between the device and memory.

If you select data channel I/O, you can select one of the sixteen maps. The symbols ?UDDA through ?UDDP correspond to maps A through P. Add the correct map to the value corresponding to the first acceptable map slot you want the operating system to use. For example, the specification ?UDDA+5 tells the operating system to use map A (?UDDA) and to start with slot 5 in that map, if possible. Offset ?UDNS tells the operating system how many contiguous map slots you will need.

If your devices use data channel maps, you should avoid using the first 128 (A – D) data channel map slots if possible. We advise this because some I/O controllers have restrictions on the number of data channel maps they can gain access to. Many older I/O controllers use a 17–, 16–, or 15–bit data channel address while newer controllers use a 19–bit data channel address. These numbers make the A through D maps more valuable than the E through P maps. We suggest that you request a data channel map starting at the highest slot the device can gain access to. If the request fails, try the next lower slot and so on. This approach leaves the lower slots available for I/O controllers that are restricted to them.

The map slots must be contiguous. Thus, if the first acceptable slot you specify is already in use, the operating system takes the next available map slot with the required number of contiguous slots.

Warning

If you issue ?IDEF system calls, you must make sure that all code and data touched by a user device interrupt handler service routine (including the handler itself) must be wired. Failure to do this will result in panics with code 14340 if a page fault occurs at interrupt level while trying to reference instructions or data.

Sample DCT

```
DCT:    .BLK    ?UDLN           ;Allocate enough space for the standard DCT.
                                ;  (DCT length = ?UDLN).
        .LOC    DCT+?UDVXM      ;TCB address of ?IMSG task.
        .DWORD  0               ;Set to 0. (The OS supplies this value.)

        .LOC    DCT+?UDVIS      ;Interrupt service routine address.
        .DWORD  INTSRV          ;Interrupt service routine address is
                                ;  INTSRV.
        .LOC    DCT+?UDVBX      ;?IXMT mailbox.
        .DWORD  0               ;Set to 0.
        .LOC    DCT+?UDDRS      ;Address of power-failure routine.
        .DWORD  -1              ;There is no power-failure routine.
        .LOC    DCT+?UDVMS      ;Interrupt service mask.
        .WORD   1B12+1B13+1B14+1B15
        .LOC    DCT+?UDRS       ;Standard/extended packet indicator.
        .WORD   0               ;Set to 0 for standard packet.
        .LOC    DCT+?UDLN       ;End of DCT.
```

User Device Termination Routine

You specify a User Device Termination Routine (UDTR) by placing a word pointer to the routine in offset ?UDDTR of the Device Control Table. (Place 0 in this offset if you don't want a UDTR.) The routine must be in Ring 4, 5, 6, or 7.

You choose a UDTR according to your application and device. After you create and load it properly the UDTR executes under the following circumstances:

- Your process issues system call ?IRMV against the device that you defined previously via system call ?IDEF or ?FIDEF.
- Your process terminates itself or a superior process terminates it, and your process previously defined one or more devices via system call ?IDEF or ?FIDEF.

Here are the rules for creating UDTRs.

- Terminate the routine with system call ?IXIT.
- Issue no other system calls in the routine.
- The routine can issue I/O instructions, such as those in the **DIx** and **DOx** families, to get the user device into the desired state.
- The initial contents of the accumulators are random; you needn't preserve them. Furthermore, it's not necessary to begin the routine with a WPSH, WSAVR, or WSSVR instruction.

Note that when you issue ?IXIT at base level and it is not in a UDTR or a user powerfail/restart routine, ?IXIT returns error ERICM.

Next is a substantial portion of a typical UDTR.

```
          ;          4180 Series Digital-to-Analog Conversion Subsystem
          ;             Termination Routine.
          ;          When a process that uses Series 4180 D/A converters
          ;             terminates we want to have zero output voltage from
          ;             both of the analog channels.  (Other applications
          ;             may need different outputs.)
          ;          The 4180 D/A subsystem device code is 23 = DACV -- a
          ;             standard MASM symbol.
DA_TERM:                          ; Beginning of the routine.
          WSUB    0,0             ; Output voltage = 0.
          WSUB    1,1             ; Analog output number 0.
          DOBC    1,DACV          ; Select an analog channel.
          DOAS    0,DACV          ; Output data goes to analog output number 0.
          WINC    1,1             ; Analog output number 1.
          DOBC    1,DACV          ; Select an analog channel.
          DOAS    0,DACV          ; Output data goes to analog output number 1.
          ?IXIT                   ; System Call.
          WBR     ERROR           ; Error return.
```

User Device Powerfail/Restart Routine

This section applies only to AOS/RT32.

You specify a User Device Powerfail/Restart Routine (UDPR) by placing a word pointer to the routine in offset ?UDDRS of the Device Control Table. (Place 0 in this offset if you don't want a UDPR.) The routine must be in Ring 4, 5, 6, or 7.

You choose a UDPR according to your application and device. After you create and load it properly the UDPR executes under the following circumstances:

- A power failure is occurring. AC1 contains 0, and the operating system transfers to the routine that offset ?UDDRS points to. This routine should execute its power failure section.
- Power is returning after a power failure. AC1 contains –1, and the operating system transfers to the routine that offset ?UDDRS points to. This routine should execute its power restart section.

Here are the rules for creating UDPRs.

- Terminate the routine with system call ?IXIT.
- Issue no other system calls in the routine.
- The routine can issue I/O instructions, such as those in the **DIx** and **DOx** families, to get the user device into the desired state.

- The initial contents of the accumulators (except AC1) are random; you needn't preserve them. Furthermore, it's not necessary to begin the routine with a WPSH, WSAVR, or WSSVR instruction.

- When you don't need one of the power failure and restart routines you still must specify an empty routine (with only system call ?IXIT) for the unneeded routine.

Note that when you issue ?IXIT at base level and it is not in a UDPR or a user device termination routine, ?IXIT returns error ERICM.

Next is a substantial portion of a typical UDPR.

```
          ;          4180 Series Digital-to-Analog Conversion Subsystem
          ;             Powerfail/Restart Routine.
          ;
          ;          When a process that uses Series 4180 D/A converters
          ;             encounters a power failure we want to have zero output
          ;             voltage from both of the analog channels.  When
          ;             this process encounters the return of power we want
          ;             to set output voltages to the default values defined
          ;             for each channel.
          ;
          ;          The 4180 D/A subsystem device code is 23 = DACV -- a
          ;             standard MASM symbol.
DA_POWER:                      ; UDPR entry point from the system.
          WSEQ   1,1           ; Is it power down?
          WBR    POWER_UP      ; No.
          ;
          ; Power is going down.  All we want to do is to set the output
          ;    of both D/A converters to zero.
POWER_DOWN:                    ; Beginning of the power failure routine.
          WSUB   0,0           ; Output voltage = 0.
          WSUB   1,1           ; Analog output number 0.
          DOBC   1,DACV        ; Select an analog channel.
          DOAS   0,DACV        ; Output data goes to analog output number 0.
          WINC   1,1           ; Analog output number 1.
          DOBC   1,DACV        ; Select an analog channel.
          DOAS   0,DACV        ; Output data goes to analog output number 1.
          WBR    DONE          ; Exit from the power failure routine.
          ;
          ; Power is back.
POWER_UP:                      ; Beginning of the power restart routine.
                               ;    Set the default outputs for both channels.
          XNLDA  0,DEF1        ; Start with the default value of channel 1.
          WSUB   1,1           ; Analog output number 0.
          DOBC   1,DACV        ; Select an analog channel.
          DOAS   0,DACV        ; Output data goes to analog output number 0.
          XNLDA  0,DEF2        ; Start with the default value of channel 2.
          WINC   1,1           ; Analog output number 1.
          DOBC   1,DACV        ; Select an analog channel.
          DOAS   0,DACV        ; Output data goes to analog output number 1.
DONE:     ?IXIT                ; End of the device routine.
          WBR    ERROR         ; Error return.
          ;
          ;          The output voltage range is 0 - 10 volts.  The converters
          ;             have 12 bits of precision.  Let the default value for
          ;             for channel 1 be 1 volt and the default value for
          ;             channel 2 be 2 volts.
DEF1:     .WORD  4096./10.     ; The default for channel 1 is 1 volt.
DEF2:     .WORD  4096./5.      ; The default for channel 2 is 2 volts.
```

Notes

- See the descriptions of ?FIDEF, ?IXMT, and ?STMAP in this chapter.

?IDGOTO
error return
normal return

Input

AC0	Address of the new code sequence
AC1	TID of the target task
AC2	Reserved (Set to 0.)

Output

AC0	Unchanged
AC1	Unchanged
AC2	Undefined

Error Codes in AC0

ERTID	Task ID error (The OS could not find a matching TID.)
ERVWP	Invalid word pointer passed as a system call argument

Why Use It?

?IDGOTO allows you to redirect a task that is currently executing, without having to kill the task and re–initiate it.

Because it will lift certain task suspensions, you can use ?IDGOTO to provide interrupt processing after a Ctrl–C Ctrl–A terminal interrupt. For example, you might create an interrupt task (via ?INTWT) to monitor your terminal for Ctrl–C Ctrl–A sequences. In this case, if the interrupt task were to detect an interrupt, it would issue ?IDGOTO to redirect the main task.

Who Can Use It?

There are no special process privileges needed to issue this call, and there are no restrictions concerning file access.

What It Does

?IDGOTO redirects the task that you specify in AC1 to a new code sequence, where it resumes executing the next time it gains CPU control. If the target task has any outstanding system calls, ?IDGOTO aborts those calls and readies the task. ?IDGOTO also readies the target task if it explicitly suspended itself with ?SUS, if it became suspended after issuing ?REC or ?XMTW, or if another task suspended it with ?IDSUS or ?PRSUS. The target task's priority remains the same after an ?IDGOTO.

Before you issue ?IDGOTO, load AC0 with the address of the new code sequence, and load AC1 with the TID of the target task. Note that a task may not use ?IDGOTO to redirect itself; thus, the TID that you specify in AC1 cannot be that of the calling task.

Notes

- See the descriptions of ?SUS, ?REC, ?XMTW, ?IDSUS, or ?PRSUS in this chapter.

?IDKIL
error return
normal return

Input

AC0	Reserved (Set to 0.)
AC1	TID of the task you want to kill
AC2	Reserved (Set to 0.)

Output

AC0	Undefined
AC1	Unchanged
AC2	Undefined

Error Codes in AC0

ERTID	Task ID error (The OS could not find a matching TID.)

Why Use It?

Like the other system calls that require a task ID, ?IDKIL allows you to perform an action — in this case a "kill" — on a specific target task, rather than on all tasks of a given priority. Thus, you can use ?IDKIL to kill a task without affecting other tasks of the same priority.

Who Can Use It?

There are no special process privileges needed to issue this call, and there are no restrictions concerning file access.

What It Does

?IDKIL terminates the task that you specify in AC1. If the target task is suspended because of an outstanding system call, the operating system readies the target task by aborting the outstanding system call.

If you supplied a kill–processing routine for the target task with ?KILAD, the operating system passes control to that routine as it executes ?IDKIL. The kill–processing routine should end with a ?KILL system call. ?KILL, in turn, invokes a ?UKIL task–termination routine — either the system default ?UKIL or a ?UKIL you have defined. ?UKIL then terminates the task.

If you did not provide a kill–processing routine, control passes to the appropriate ?UKIL immediately.

Notes

- See the descriptions of ?KILAD, ?IDKIL, and ?KILL in this chapter.

?IDPRI
error return
normal return

Input		Output	
AC0	New priority for the target task	AC0	Unchanged
AC1	TID of the target task	AC1	Unchanged
AC2	Reserved (Set to 0.)	AC2	Undefined

Error Codes in AC0

ERTID Task ID error (The OS could not find a matching TID.)

Why Use It?

?IDPRI is analogous to ?PRIPR, which changes the priority of a process. Both system calls allow you to influence the operating system scheduling activities by adjusting the priority numbers of a process or a task.

You can use ?IDPRI to reschedule tasks immediately. For example, you can issue ?IDPRI to give a task a higher relative priority than others for CPU control, or to give a task a lower priority (which upgrades the relative priorities of other tasks).

Who Can Use It?

There are no special process privileges needed to issue this call, and there are no restrictions concerning file access.

What It Does

?IDPRI changes the priority of the task that you specify in AC1. Before you issue ?IDPRI, load AC1 with the TID of the target task and load AC0 with the priority number you wish to assign. You can use ?IDPRI to change the priority of the calling task.

When you use ?IDPRI to assign a task a higher priority than the current ready task, the operating system reschedules the other task before it returns control to the ?IDPRI caller.

Notes

- See the descriptions of ?PRIPR and ?PRI in this chapter.

?IDRDY
error return
normal return

Input		Output	
AC0	Reserved (Set to 0.)	AC0	Undefined
AC1	TID of the target task	AC1	Unchanged
AC2	Reserved (Set to 0.)	AC2	Undefined

Error Codes in AC0

ERTID Task ID error (The OS could not find a matching TID.)

Why Use It?

You can use ?IDRDY to revoke a task's suspension and, thus, ready the task for scheduling and eventual execution. Like the other tasking system calls that require a TID, ?IDRDY allows you to ready a single task, rather than all tasks of a given priority.

You can only issue system calls that require a TID against tasks that have nonzero TIDs. Thus, to ready a task without a TID, you must use ?PRRDY, not ?IDRDY.

Who Can Use It?

There are no special process privileges needed to issue this call, and there are no restrictions concerning file access.

What It Does

?IDRDY readies the task that you specify in AC1, but only if it was explicitly suspended by ?SUS, ?IDSUS, or ?PRSUS.

When you use ?IDRDY to ready a task that has a lower priority than the calling task, the operating system reschedules the tasks.

Notes

- See the descriptions of ?SUS, ?IDSUS, ?PRSUS, and ?PRRDY in this chapter.

?IDSTAT Returns task status word (16–bit processes only).

?IDSTAT
error return
normal return

Input

AC0	Reserved (Set to 0.)
AC1	TID of the target task
AC2	Reserved (Set to 0.)

Output

AC0	Status word
AC1	Unchanged
AC2	Contains –1

Error Codes in AC0

ERTID	Task ID error (The OS cannot find the task whose TID you specified.)
ERICM	32–bit program incorrectly attempted to call ?IDSTAT

Why Use It?

?IDSTAT allows you to determine the status of a particular task.

Who Can Use It?

There are no special process privileges needed to issue this call, and there are no restrictions concerning file access.

What It Does

?IDSTAT returns a word that describes the target task's status; that is, the ?TSTAT word of the task's task control block (TCB).

?IDSUS
error return
normal return

Input

AC0	Reserved (Set to 0.)
AC1	TID of the target task
AC2	Reserved (Set to 0.)

Output

AC0	Undefined
AC1	Unchanged
AC2	Undefined

Error Codes in AC0

ERTID	Task ID error (The OS could not find a matching TID.)

Why Use It?

Because ?IDSUS takes a TID as an input parameter, you can use it to explicitly suspend a single task, rather than all tasks of a given priority level. Also, a task can issue ?IDSUS to suspend itself.

Who Can Use It?

There are no special process privileges needed to issue this call, and there are no restrictions concerning file access.

What It Does

?IDSUS, which is the inverse of ?IDRDY, suspends the task that you specify in AC1. Note that you can only use ?IDSUS and the other system calls that require a TID against tasks that have a TID.

Notes

- See the description of ?IDRDY in this chapter.

?IESS — Initializes an extended state save (ESS) area (16–bit processes only).

?IESS
normal return

Input		Output	
AC0	Size of the ESS area	AC0	Unchanged
AC1	Page 0 starting address	AC1	Unchanged
AC2	Starting address of the ESS area	AC2	Unchanged

Error Codes in AC0

No error codes are currently defined.

Why Use It?

?IESS allows you to store any information that you consider relevant to a task.

Who Can Use It?

There are no special process privileges needed to issue this call, and there are no restrictions concerning file access.

What It Does

?IESS initializes an extended state save (ESS) area in the unshared portion of your logical address space. The ESS area is generally used to hold task–specific information for a 16–bit process, such as the value of the program counter and its carry bit, and the current contents of the accumulators. However, you can use it for other purposes.

Each task that uses the ESS area must issue ?IESS immediately upon entering the task code; otherwise, the ESS information may be invalid.

Input to ?IESS includes the starting address of the ESS area (in the unshared area of your logical address space), and a pointer to a block of Page 0 locations in your logical address space. When the operating system schedules a 16–bit task it copies the ESS information to the designated Page 0 area. When rescheduling occurs, the operating system transfers the ESS information back to the starting address you specified in AC2 when you first issued ?IESS.

?IFPU
error return
normal return

Input

AC0	Starting address of a 2–word block that initializes the state of the floating–point status register
AC1	Reserved (Set to 0.)
AC2	Reserved (Set to 0.)

Output

AC0	Unchanged
AC1	Undefined
AC2	Undefined

Error Codes in AC0

No error codes are currently defined.

Why Use It?

Each task must issue ?IFPU before it issues floating–point instructions. This guarantees that every time the operating system gives control of the CPU to a task, the floating–point state is restored to the state the task was last in. If each task (even if your program only contains one task) does not issue ?IFPU first, your floating–point arithmetic results may not be correct.

Who Can Use It?

There are no special process privileges needed to issue this call, and there are no restrictions concerning file access.

What It Does

?IFPU uses the 2–word area pointed to by AC0 to initialize the state of the floating–point status register and to set the floating–point accumulators to zero. We recommend that you place zeros in this 2–word area.

?IHIST — Starts a histogram for a 16–bit process (16–bit processes only).

?IHIST
error return
normal return

Input

AC0	One of the following: • Byte pointer to the name of the target process • PID of the target process • –1 to start a histogram for the calling process	
AC1	One of the following: • –1 if AC0 contains a byte pointer • Any other value if AC0 contains –1 or a PID	
AC2	Address of the histogram packet	

Output

AC0	Unchanged
AC1	Unchanged
AC2	Unchanged

Error Codes in AC0

ERHIS	Error on histogram initiation/deletion
ERMPR	System call parameter address error
ERPRH	Attempt to access process not in hierarchy
ERPRV	Caller not privileged for this action
ERVBP	Invalid byte pointer passed as a system call argument

Why Use It?

?IHIST allows a 16–bit resident process to monitor a range of addresses in its own or another process's logical address space. The operating system returns an array that compares the CPU time of the target process with the CPU time used by other processes, including the operating system. ?IHIST is a useful way of obtaining a global view of CPU activity.

Who Can Use It?

There are no special process privileges needed to issue this call, but the calling process must be resident. There are no restrictions concerning file access.

What It Does

?IHIST, which is the 16–bit counterpart of ?WHIST, starts a histogram for a range of logical addresses in the target process. The calling process cannot activate more than one histogram at a time.

Before you issue ?IHIST, load AC0 with the PID of the target process or a byte pointer to its name, or with –1 to start a histogram for the calling process. Your input to AC1 depends on your input to AC0.

You must also set up a histogram packet of ?HWLTH words in your logical address space, and reserve a buffer to receive the histogram statistics. Load the packet address into AC2. (An offset within the packet points to the buffer address.) Figure 2–80 shows the structure of the histogram packet for 16–bit processes.

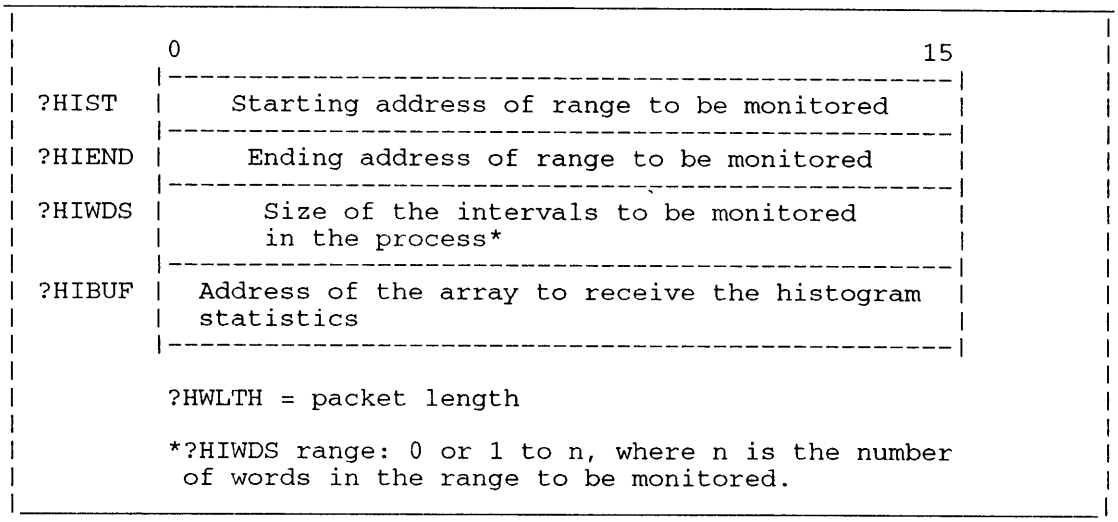


Figure 2–80. Structure of ?IHIST Packet

To start a simple histogram, which merely records how often the target process gained CPU control, set offset ?HIWDS to 0. This directs the operating system to ignore the contents of offsets ?HIST and ?HIEND and prevents the operating system from gathering range statistics. Set offset ?HIBUF to the address of the array you have reserved in your logical address space for the histogram statistics.

Each histogram array has two parts: a fixed–length header followed by double–precision array elements, which correspond to each interval that you want to monitor. Table 2–58 shows the structure of the histogram array for 16–bit processes.

Table 2–58. Structure of ?IHIST Array

Array Offset		Contents
?HTTH	?HTTL	Total number of real-time clock pulses (ticks) counted in this histogram.
?HPRH	?HPRL	Total number of ticks when the program counter (PC) was within the target process, but outside the specified range.
?HAPH	?HAPL	Total number of ticks in other processes.
?HSBH	?HSBL	Total number of ticks in the OS, except those recorded when it was in an idle loop.
?HSIH	?HSIL	Total number of ticks in a system idle loop.
?HARAY	?HARAY+1	Total number of ticks in the first interval.
...		
?HARAY+n*2-2	?HARAY+n*2-1	Total number of ticks in the nth interval.

?ILKUP
error return
normal return

Input

AC0	Byte pointer to the pathname of the target IPC entry
AC1	Reserved (Set to 0.)
AC2	Reserved (Set to 0.)

Output

AC0	Unchanged
AC1	Global port number associated with the IPC entry
AC2	File/device type of the IPC file entry

Error Codes in AC0

ERFDE File does not exist
ERIFT Illegal file type (The file that you specified in AC1 is not an IPC file.)
ERVBP Invalid byte pointer passed as a system call argument
ER_FS_OBSOLETE_IPC_FILE_DETECTED
Obsolete IPC file type has been detected; file has been deleted (AOS/VS II only)
ER_FS_DIRECTORY_NOT_AVAILABLE
Directory not available because the LDU was force released (AOS/VS II only)

Why Use It?

?ILKUP returns the global port number of the IPC file that the target process previously created. Because a process must have the correct global port number to send an IPC message to another process, ?ILKUP can be a useful preliminary step to an ?ISEND or an ?IREC.

Who Can Use It?

There are no special process privileges needed to issue this call, and there are no restrictions concerning file access.

What It Does

?ILKUP returns the global port number (which includes the PID, the ring number, and the local port number of the creator) associated with an IPC file entry that you specify. If the file does not exist or is not an IPC file, ?ILKUP fails and the operating system returns error code ERFDE or ERIFT in AC0.

Note that if you specify an IPC file whose local port number is 0, the number returned in AC1 is the port number of your terminal, not the port number of the file that you specified.

Notes

- See the descriptions of ?IREC and ?ISEND in this chapter.
- See PARU.32.SR or PARU.16.SR, for the IPC file and device types.

?IMERGE
error return
normal return

Input

AC0 One of the following:

- Global port number (32–bit users only)
- OPH, which indicates high–order bits of global port number (16–bit users only)

AC1 One of the following:

- Reserved (Set to 0.) (32–bit users only)
- OPL, which indicates low–order bits of global port number (16–bit users only)

AC2 Ring field in Bits 29 through 31

Output

AC0 One of the following:

- Global port number, including the ring field (32–bit users only)
- OPH, which indicates high–order bits of global port number including the ring field (16–bit users only)

AC1 One of the following:

- Undefined (32–bit users only)
- OPL, which indicates low–order bits of global port number, including ring field (16–bit users only)

AC2 Unchanged

Error Codes in AC0

No error codes are currently defined.

Why Use It?

?IMERGE allows both 16– and 32–bit users to modify the ring field within a global port so that it can receive an IPC message in an alternate ring.

Who Can Use It?

There are no special process privileges needed to issue this call, and there are no restrictions concerning file access.

What It Does

?IMERGE incorporates the new ring field that you specified in AC2 into the process's global port number.

Notes

- See the description of ?ISPLIT in this chapter.

?IMSG
error return
normal return

Input

AC0	Reserved (Set to 0.)
AC1	Reserved (Set to 0.)
AC2	User DCT address of the sending device

Output

AC0	Undefined
AC1	Interrupt service message
AC2	Unchanged

Error Codes in AC0

ERMIM	Multiple ?IMSG calls to same DCT (Only one task can wait for an interrupt service message at a time)
ERVWP	Invalid word pointer passed as a system call argument

Why Use It?

?IMSG is the "receive" counterpart of ?IXMT. ?IXMT and ?IMSG allow you to pass data from an interrupt service routine to an outside task, or to synchronize the two tasks.

Once a process has entered an interrupt service routine, ?FIXMT, ?IXIT, (exit from the routine), ?IXMT, and ?SIGNL are the only system calls that process can issue.

Who Can Use It?

There are no special process privileges needed to issue this call beyond those that ?IDEF requires, and there are no restrictions concerning file access.

What It Does

?IMSG lets the calling task receive a message sent by an interrupt service routine with ?IXMT. ?IXMT and ?IMSG can be in any order. If the receiving task issues the ?IMSG before the sender issues the ?IXMT, the operating system suspends the receiving task pending the transmission. As soon as the sending interrupt service routine issues ?IXMT, the receiver becomes active again, provided no other tasks suspended it in the meantime.

Before you can use ?IXMT and ?IMSG, you must use ?IDEF to define the sending interrupt service routine. Also, before you issue ?IMSG, you must load AC2 with the DCT address of the device that is associated with the sending interrupt service routine. When the operating system executes the sender's ?IXMT, it passes the message to the mailbox field that it has already set aside for the device. When the operating system executes ?IMSG, it moves the message from the mailbox to AC1.

You must define the user DCT address of the sending device, using ?IDEF, before you issue ?IMSG.

Notes

- See the descriptions of ?IXMT, ?IXIT, and ?IDEF in this chapter.

AOS/VS

?INIT *[packet address]*
error return
normal return

Input

AC0	One of the following: • Byte pointer to a 32–byte receive buffer for the LD name • 0 if there is no buffer
AC1	One of the following: • –1 to graft the new LD to the system root directory (:) • 0 to graft the LD to the caller's working directory • Byte pointer to the pathname of the target directory
AC2	Address of the ?INIT packet, unless you specify the address as an argument to ?INIT

Output

AC0	Unchanged
AC1	Unchanged
AC2	Address of the ?INIT packet

Error Codes in AC0

EREAD	Execute access denied
ERFIX	Can't initialize LD, run FIXUP on it
ERIDD	Inconsistent DIB (disk indentification block) data (Disk is not a valid operating system disk type.)
ERIDU	Incomplete LD (logical disk)
ERILD	Inconsistent LD
ERMIS	Disk and file system revision numbers don't match
ERVBP	Invalid byte pointer passed as a system call argument
ERVWP	Invalid word pointer passed as a system call argument

Why Use It?

Before you can use a logical disk, you must initialize it. You can do this with the CLI command INITIALIZE, or with the ?INIT system call. (You tell the operating system that the controller for the logical disk exists during the system–generation procedure.) Initializing a logical disk allows you to incorporate its files into the existing file structure.

?INIT Continued

Who Can Use It?

There are no special process privileges needed to issue this call. To use ?INIT, the calling process must have Owner access to the target LD root and either Write or Append access to the target directory. In addition, the caller must have Execute access to each disk unit in the LD and Execute access to the target directory.

What It Does

?INIT initializes a logical disk and grafts the LD's local directory structure to the directory that you specify in AC1 (that is, the system root, the current working directory, or another directory).

?INIT requires a packet of 2n + 1 words, where n is the number of disk units that make up the logical disk you are initializing. Table 2–59 lists the contents of the ?INIT packet.

Table 2–59. Contents of ?INIT Packet*

Offset	Contents
0	Number of physical units in the LD.
1 and 2 (doubleword)	Byte pointer to the name of the first disk in the LD.
3 and 4 (doubleword)	Byte pointer to the name of the second disk in the LD.
5 and 6 (doubleword)	Byte pointer to the name of the third disk in the LD.
...	
2*n-1 and 2*n (doubleword)	Byte pointer to the name of the nth disk in the LD.

* There is no default unless otherwise specified.

You must terminate each disk name with a null byte.

Notes

- Refer to the manual *Installing, Starting, and Stopping AOS/VS II* for a list of the valid disk names and for information on the system–generation procedure.

?INTWT — Defines a terminal interrupt task.

?INTWT
error return
normal return

Input

None

Output

None

Error Codes in AC0

No error codes are currently defined.

Why Use It?

The operating system enables terminal interrupts when a program begins to execute. Before the operating system will recognize a terminal interrupt, however, you must define a terminal–interrupt task with ?INTWT. Because the operating system disables terminal interrupts after it executes the interrupt task; you must issue another ?INTWT each time you want to re–enable terminal interrupts.

Who Can Use It?

There are no special process privileges needed to issue this call, and there are no restrictions concerning file access.

What It Does

?INTWT defines a task to handle a terminal interrupt (that is, a Ctrl–C Ctrl–A control sequence, which suspends the process terminal's output). In addition, ?INTWT re–enables terminal interrupts if they were previously disabled by a terminal interrupt.

When the operating system detects a terminal interrupt, it passes control to the task defined by ?INTWT, and then disables further terminal interrupts until you issue either another ?INTWT or a ?OEBL. You should initiate the terminal–interrupt task at the highest priority so that it will receive control as soon as possible to service the interrupt.

Notes

- See the description of ?OEBL in this chapter.
- See the description of ?KWAIT in this chapter.

?IQTSK
error return
normal return

Input		Output	
AC0	Priority level for the queued task manager (0 to assign it the caller's priority level)	AC0	Unchanged
AC1	TID for the queued task manager	AC1	Unchanged
AC2	Reserved (Set to 0.)	AC2	Undefined

Error Codes in AC0

ERQTS Error in queued task request (An active queued task manager already exists.)

Why Use It?

You must issue ?IQTSK before you can issue ?TASK to create queued tasks. Queued tasks allow your program to delay task initiation and/or direct the operating system to reinitiate tasks at specific time intervals.

Who Can Use It?

There are no special process privileges needed to issue this call, and there are no restrictions concerning file access.

What It Does

?IQTSK creates a queued task manager task to manage tasks you initiate with the ?TASK queued task creation facility. You can create only one queued task manager. Therefore, you can issue only one ?IQTSK per program. However, when you issue ?IQTSK, it creates a task. Therefore, you must allow for one extra task control block (TCB). For example, if you create three tasks with one ?TASK system call, which you want queued, the operating system is using five TCBs.

Before you issue ?IQTSK, load AC0 with a priority level for the queued task manager, and load AC1 with its task identifier (TID). You must specify a unique task ID, because the operating system needs this information to control the task manager. If you specify 0 in AC0, the operating system assigns the queued task manager the same priority as the calling task.

When you terminate the last active task in a process, the operating system terminates the queued task manager, provided it is still active and has an empty task initiation queue.

Notes

- See the descriptions of ?TASK and ?DQTASK in this chapter.
- Don't change the system time after issuing ?IQTSK. Unpredictable results will occur.

?IREC *[header address]*
error return
normal return

Input

AC0	Reserved (Set to 0.)
AC1	Reserved (Set to 0.)
AC2	Address of the IPC header, unless you specify the address as an argument to ?IREC

Output

AC0	Undefined
AC1	Undefined
AC2	Address of the IPC header

Error Codes in AC0

ERMPR	System call parameter address error (The receive buffer is not in the unshared area of your address space.)
ERVBP	Invalid byte pointer passed as a system call argument
ERVWP	Invalid word pointer passed as a system call argument

Why Use It?

You would issue ?IREC either to receive an IPC message that another process sent via ?ISEND or to receive a process termination or connection management message.

Who Can Use It?

There are no special process privileges needed to issue this call, and there are no restrictions concerning file access.

What It Does

?IREC opens a receiving port for the calling process, which allows the caller to receive an IPC message from a sending process. ?IREC requires a header, which lists, among other information, the origin and destination ports for the message.

Before you issue ?IREC, set up the header and reserve a receive buffer in your logical address space. You can cite the header address as an argument to ?IREC or load it into AC2 before you issue ?IREC. The ?IREC header consists of ?IPLTH words. Figure 2–81 shows the header's structure, and Table 2–60 describes its contents. Table 2–60 also describes the optional contents of ?ISFL in the ?IREC header.

Offset ?IOPH contains the global port number of the calling process (issue ?ILKUP to obtain this). If you set offset ?IOPH to 0, the calling process can accept messages from any sender. Similarly, if you set offset ?IDPN (the destination port number) to 0, the calling process can accept the message on any of its ports. During message transmission, the operating system writes the actual origin port number into offset ?IOPH, and the actual destination port number into offset ?IDPN.

Offset ?IPTR points to the receive buffer, which must be in the unshared area of your logical address space. (See Table 2–60 which describes the system flag words and user flag words.)

By default, the operating system suspends the receiver if there is no outstanding message for its ?IREC. You can avoid this by setting flag ?IFNBK in offset ?ISFL of the receive header. This flag signals the operating system to return an error to the receiver if there is no message.

An ?IREC from a particular ring can only receive an IPC message whose destination is that particular ring.

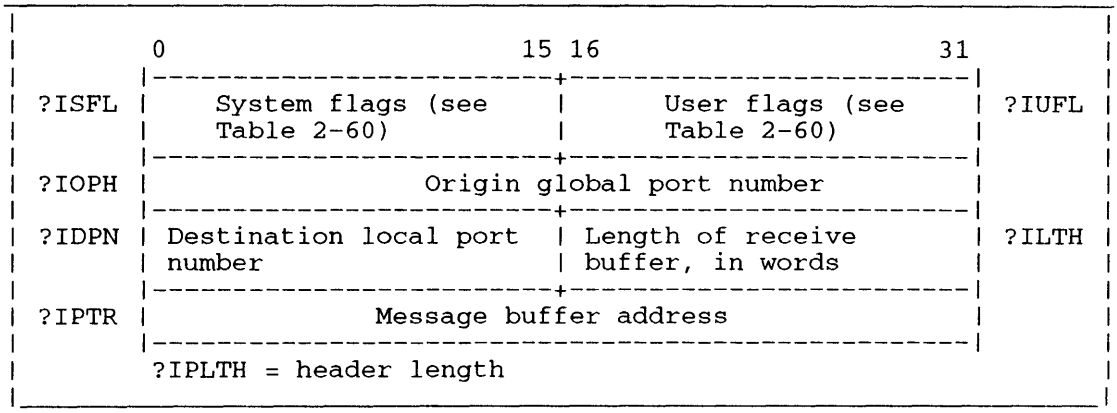


Figure 2–81. Structure of ?IREC Header

Table 2–60. Contents of ?IREC Header*

Offset	Contents
?ISFL	System flag word ?IFRFM--Receive a looped message (sent by this process to itself). Use this to test a process to see if both IPCs are working properly. Also, this is a good way for one program to communicate with another program in a single process (?CHAIN). ?IFSOV--Spool the message if the receive buffer is too small. ?IFNBK--Signal an error if there is no spooled message for this receiver. ?IFRING--Contains the sender's ring field (returned by the OS).(The ring field is a 3-bit field, Bits 13 through 15.) ?IFPR---Indicates .PR file type of sender; 0 if the sender is a 32-bit process, or 1 if the sender is a 16-bit process (returned by the OS).
?IUFL	User flag word. (See Figure 2-82.)
?IOPH (doubleword)	Origin global port number.
?IDPN	Destination local port number.
?ILTH	Size of the receive buffer, in words.
?IPTR (doubleword)	Address of the receive buffer.

* There is no default unless otherwise specified.

After ?IREC has finished you can identify the process that issued the associated ?ISEND or ?IS.R call. To do this, load the origin global port number from offset ?IOPH of the ?IREC packet into AC0 and then issue ?ISPLIT. After ?ISPLIT finishes AC1 will contain the PID of the sending process.

Sample Packet

```
HDR:    .BLK    ?IPLTH          ;Allocate enough space for the
                                ;packet.  Packet length = ?IPLTH.
        .LOC    HDR+?ISFL       ;System flags.
        .WORD   ?IFSOV          ;If buffer is too short, spool.
        .LOC    HDR+?IUFL       ;User flags.
        .WORD   0               ;Set to 0.
        .LOC    HDR+?IOPH       ;Global port number of calling
        .DWORD  0               ;process.  (Set to 0 to accept
                                ;messages from any sender.)
        .LOC    HDR+?IDPN       ;Destination port number.
        .WORD   0               ;Set to 0 to accept messages on any
                                ;of the calling process's ports.
        .LOC    HDR+?ILTH       ;Receive buffer length (words).
        .WORD   80./2           ;Buffer length is 80./2.  Reset
                                ;buffer length after each ?IREC.
        .LOC    HDR+?IPTR       ;Receive buffer address (must be in
        .DWORD  RECBUF          ;unshared area of your logical
                                ;address space).
        .LOC    HDR+?IPLTH      ;End of ?IREC header.
```

Process Termination Messages

When a process terminates, the system sends a process termination IPC message to the father process (who created the terminating process). The system also sends connection management messages to the server (or customer) if the status of the connection in a customer/server relationship is affected. For a process to receive a process termination or connection management message, it must issue an ?IREC system call with offset ?IOPH set to a special system port, ?SPTM.

The location of the PID number, that ?IREC receives as part of a process termination or connection management message, depends on two things: the type of process that the receiver (i.e., the process that issues ?IREC) is, and the type of message that is received, as follows:

- A–type (smallPID) process, process termination message — the PID is in the right byte of offset ?IUFL. If ?ILTH contains 0, then the contents of ?IPTR are undefined. If ?ILTH doesn't contain 0, then ?IPTR contains a word pointer to a process termination message as described in the sections "Termination Messages for A–Type 32–bit Processes," and "Termination Messages for A–Type 16–bit Processes." For more information about process types see *Managing AOS/VS and AOS/VS II* (093–000541), and *AOS/VS System Concepts* (093–000335).

- B– or C–type (hybrid or anyPID programs) process, process termination message — the operating system returns 0 in the right byte of offset ?IUFL. If ?ILTH contains 0, then the contents of ?IPTR are undefined. If ?ILTH doesn't contain 0, then ?IPTR contains a word pointer to a process termination message as described in the section "Termination Messages for B–Type and C–Type Processes." For more information about process types see *Managing AOS/VS and AOS/VS II* (093–000541), and *AOS/VS System Concepts* (093–000335).

- A–type process, connection management (obituary) message — the PID is in the right byte of offset ?IUFL. Regardless of the contents of ?ILTH, the 24 high order bits of ?IPTR are undefined and the low order 8 bits contain a bit map according to the description of connection management in the *AOS/VS System Concepts* manual.

- B– or C–type process, connection management (obituary) message — the PID is in the 16 low order bits of offset ?IPTR (and the 16 high order bits are undefined). The contents of offset ?ILTH are unimportant, but the right byte of offset ?IUFL contains a bit map according to the description of connection management in the *AOS/VS System Concepts* manual.

An A–type process receives a termination message as defined in one of the following two places:

- Table 2–60 (page 2–296) and Table 2–63.1 (page 2–303)
- Table 2–60 (page 2–296) and Table 2–63 (page 2–301)

A B–type or C–type process receives a termination message whose receive buffers are defined in one of the following two places:

- Table 2–63.2 (page 2–304.2) in the section "Termination Messages for B–Type and C–Type Processes," is for 32–bit processes.
- Table 2–64 (page 2–304.6) in the section "Termination Messages for B–Type and C–Type Processes," is for 16–bit processes.

The operating system uses offset ?IUFL of the receive header to describe the reason for the Process Termination Message and to identify the process on whose behalf it is sending the message. Figure 2–82 shows the structure of offset ?IUFL.

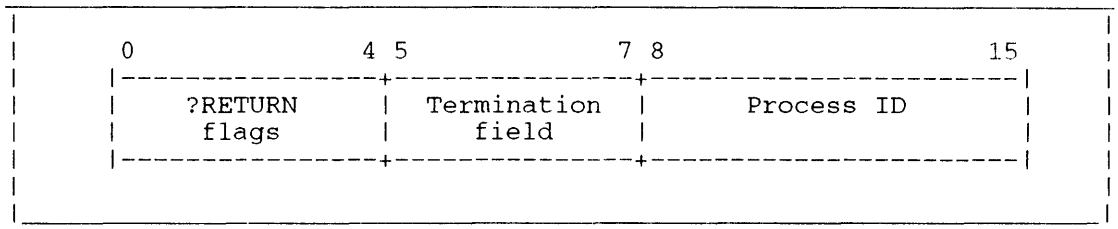


Figure 2–82. Structure of Offset ?IUFL

The termination field (bits 5 through 7) in offset ?IUFL is for the codes that the operating system uses to indicate why the termination message is being sent. The low order byte of offset ?IUFL always contains the PID of the process from which the message originated.

The termination field may contain any of the codes listed in Table 2–62, depending on the reason for the message. Note that when the termination field holds the value ?TEXT, you can find the real cause for the message in the first word of the message itself.

If a ?RETURN system call causes the termination message, the operating system uses the ?RETURN flags (bits 0 through 4) of the ?IUFL offset to provide additional information about the message format. See Table 2–61 for the ?RETURN flags.

If the message is in standard CLI format, the return flag is set to ?RFCF, and the remaining flags (?RFEC, ?RFWA, ?RFER, and ?RFAB) have their conventional meanings. If the father is not the CLI, or if ?RFCF is not set, the father and son must agree about the contents of the message.

Table 2–61. Process Termination Codes in the ?RETURN Portion of ?IUFL

Code	Meaning
?RFCF	The termination message is in CLI format (the CLI is the father).
?RFEC	AC0 contains the error code.
?RFWA	A warning condition caused the termination.
?RFER	An error condition caused the termination.
?RFAB	An abort condition caused the termination.

If a Ctrl–C Ctrl–B (or Ctrl–C Ctrl–E) interrupt sequence caused the process to terminate, ?TCIN appears in the termination field of ?IUFL. If the operating system terminated the process because of an error condition, the system returns ?TAOS as the cause of termination. You can find a further explanation for the termination by looking at the error code returned in the ?IPTR offset of the message header.

If a customer/server relationship caused the message, the system returns codes ?TBCX, ?TCCX, or ?TABR in the termination field.

Table 2–62. Process Termination Codes in Offset ?IUFL for ?IREC and ?ISEND Headers

Code	Meaning
?TSELF	Either a 16-bit process terminated itself with a ?TERM or a ?RETURN system call or a 32-bit process terminated itself with a ?TERM system call.
?TRAP	A user trap terminated a 16-bit process; Word 5 of the IPC message to the father describes the trap.
?TCIN	An abort terminal interrupt (Ctrl-C Ctrl-B sequence) terminated a process.
?TSUP	Terminated by a superior process.
?TAOS	The OS terminated a process because of an error; offset ?IPTR in the IPC header contains the error code.
?TBCX	A process broke a connection that was established via the connection-management system calls.
?TCCX	The connection still exists, but the process chained.
?TEXT	Indicates an extended termination code; the extended code appears in offset 0 (first word) of the IPC message. A termination code of ?TEXT means that the actual termination code is a 16-bit code in the first word of the termination message in the receive buffer. The The following list describes these extended termination codes.

Extended Code	Meaning
?T32T	A 32-bit process terminated itself with a ?TERM or a ?RETURN system call.
?TR32	A 32-bit process terminated because of a user trap; word 10 of the termination message describes the trap.
?TABR	Task abort notification to a server process. This involves customer/server relationship, ?IDGOTO, ?IS.R, and ?IREC, either 16-bit or 32-bit process.

Termination Messages for A–Type 32–Bit Processes

When a 32–bit process terminates because of a ?TERM (without the optional message) or ?RETURN system call, the operating system sets the termination field of ?IUFL to ?TEXT and sets the first word of the termination message to ?T32T. If the message is due to a ?RETURN, the operating system uses the ?RETURN flags field of ?IUFL, and the message has the following format:

Word 0	?T32T (the extended termination code)
Word 1	Byte length of the ?RETURN message text
Words 2 and 3	Error code from ?RETURN AC0
Word 4	Start of message text

If the process terminated with a ?TERM with the optional message, the operating system forwards the message without modification.

If the 32–bit process terminated because of a user trap, the operating system sends a message in the format shown in Table 2–63. The length of this message is ?TPLN words.

Table 2–63. ?TEXT Code Termination Messages Sent on an A–Type 32–Bit Process User Trap

Word	Contents
0	?TR32 (the extended termination code).
1 and 2	AC0 contents.
3 and 4	AC1 contents.
5 and 6	AC2 contents.
7 and 8	AC3 contents.
9	Bit 0, carry; Bits 1 through 15, high-order bits of program counter.
10	Low-order bits of program counter.

(continued)

Table 2–63. ?TEXT Code Termination Messages Sent on an A–Type 32–Bit Process User Trap

Word	Contents
11	The following flag bits, which describe the trap
	Bit 0=0 Trap occurred while control was in the user context.
	Bit 0=1 Trap occurred while control was in the operating system.
	Bit 3=1 A node time-out occurred. (This is a hardware error.)
	Bit 4=1 Process tried to execute a privileged instruction.
	Bit 5=1 Process tried to return to an inner ring from a subroutine call. (This is a violation of the ring structure.)
	Bit 6=1 Process tried to issue a subroutine call to an outer ring. (This is a violation of the ring structure.)
	Bit 7=1 Gate protection error. (This is a violation of the ring structure.)
	Bit 8=1 Process tried to reference an address in an inner ring. (This is a violation of the ring structure.)
	Bit 9=1 Process tried to read a read-protected page.
	Bit 10=1 Process tried to execute data in an execute-protected area.
	Bit 12=1 Process tried to write into a write-protected area.
	Bit 13=1 Memory map validity error. (The process tried to refer to an address outside the user context.)
	Bit 14=1 Defer error. (The process tried to use more than 16 levels of indirection in an address reference.)
	Bit 15=1 Process tried to issue a machine-level I/O instruction without issuing the ?DEBL system call.

(concluded)

Termination Messages for A–Type 16–Bit Processes

When a 16–bit process terminates by issuing a ?RETURN or ?TERM system call, the system returns flag ?TSELF to the termination field in offset ?IUFL.

If the process terminated via the ?RETURN system call, the system sends one or more of the codes in Table 2–61 to the ?RETURN Flags field of ?IUFL. Furthermore, if ?RETURN was used, a 2–word header precedes the text of the message furnished by the ?RETURN:

Word 0 Length of the ?RETURN message in bytes.
Word 1 The error code (if any) found in ?RETURN's AC0.

The text of the message follows this header. If the process supplied no message, only the first two words appear.

If the process terminated itself via the ?TERM system call, the operating system returns either the termination message specified by the system call or returns nothing to the ?IREC receive buffer, if no message was sent.

If the 16–bit process terminated because of a user trap, the operating system sets the termination field to ?TRAP and sends the father a six–word message, as shown in Table 2–63.1.

If the process terminated because of an abort terminal interrupt (Ctrl–C Ctrl–B) or a ?TERM issued by a superior process, the operating system returns ?TCIN or ?TSUP, respectively, to the termination field, but sends no message.

Table 2–63.1. ?TRAP Termination Messages for A–Type 16–Bit Processes

Word	Contents
0	AC0 contents at the time of the trap.
1	AC1 contents at the time of the trap.
2	AC2 contents at the time of the trap.
3	AC3 contents at the time of the trap.
4	Bit 0, carry; Bits 1 through 15, program counter value.
5	The following flag bits, which describe the trap: Bit 0=0 Trap occurred while control was in the user context. Bit 0=1 Trap occurred while control was in the operating system. Bit 12=1 Process tried to write into a write-protected area. Bit 13=1 Memory map validity error. (The process tried to refer to an address outside the user context.) Bit 14=1 Defer error. (The process tried to use more than 16 levels of indirection in an address reference.) Bit 15=1 Process tried to issue a machine-level I/O instruction without issuing the ?DEBL system call.

Termination Messages for B–Type and C–Type Processes

B–type or C–type processes receive the new style termination message described on the next several pages. These formats apply only to issuing processes that are type B or type C.

The calling process is either a 32–bit one or a 16–bit one. Figure 2–82.1 describes the 32–bit termination message format, and Table 2–63.2 describes its contents. Figure 2–82.2 describes the 16–bit termination message format and Table 2–64 describes its contents.

Word ?IUFL of the termination IPC header contains the value of ?TEXT in the termination field, and the value indicates an extended termination message. Also, the PID field of ?IUFL contains 0.

If the ?TERM or ?RETURN call in the terminated process has a user–supplied message, then the values in words ?ISFL and ?IUFL of the user–supplied IPC header go to offsets ?TMR8 and ?TMR9 respectively.

A ?RETURN call has the return flags set in the IPC header for a user–supplied message. The default messages from a ?RETURN call are unchanged and will appear in their entirety in the ?TMMSG area of the termination message. Any user–supplied IPC to a ?TERM call will similarly be appended in this area. If the terminated process passes only an IPC header, then offset ?TMMLG contains zero and the first two words of offset ?TMMSG contain ?IPTR. Note that user–supplied messages are only appended to the default message; they never replace it.

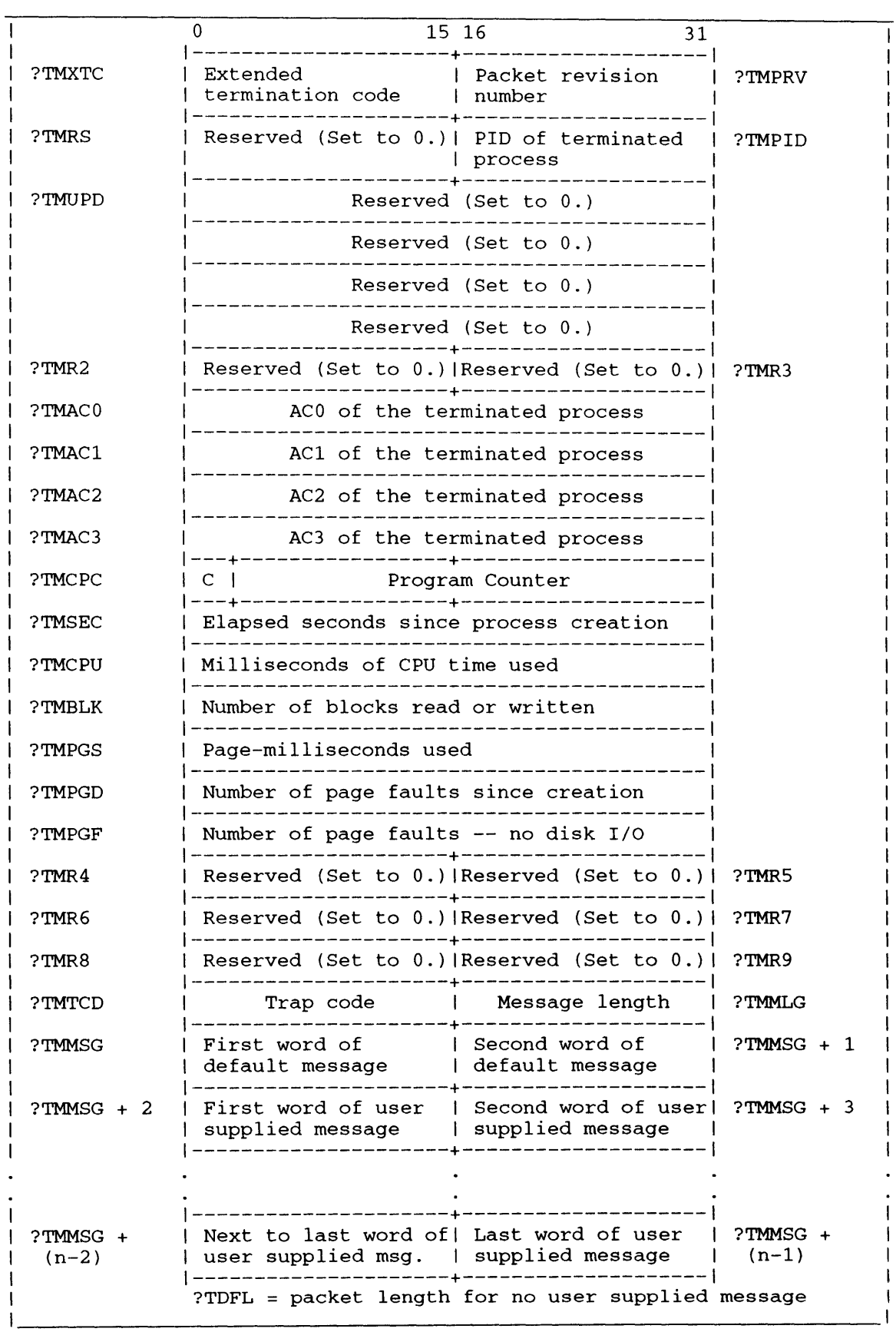


Figure 2–82.1. Structure of Termination Message from a 32–bit B– or C–Type Process

Table 2–63.2. Contents of Termination Message from a 32–bit B– or C–Type Process

Offset	Contents
?TMXTC	The operating system returns the extended termination code to indicate the type of process termination. The values and meanings of these codes are as follows. ?XT16T -- A 16-bit termination of self occurred. ?XTR16 -- A 16-bit user trap occurred. ?XTCIN -- Termination by a terminal interrupt. ?XTSUP -- Termination by a superior process. ?XTAOS -- Termination by AOS. ?XTBCX -- Customer connection broken. ?XTCCX -- Customer chained. ?XTABR -- Customer did ?TABT. ?XT32T -- A 32-bit termination of self occurred. ?XTR32 -- A 32-bit user trap occurred.
?TMPRV	Packet revision number. Place ?TM6 here.
?TMRS	Reserved. Set to 0.
?TMPID	The operating system returns the PID of the terminated process.
?TMUPD (8. words)	Reserved. Set to 0.
?TMR2	Reserved. Set to 0.
?TMR3	Reserved. Set to 0.
?TMAC0 (double-word)	The operating system returns the contents of AC0 at the time of the termination. This offset is valid only on a trap (?TMXTC contains ?XTR16 or ?XTR32). The high-order word is undefined if a 16-bit process terminated.
?TMAC1 (double-word)	The operating system returns the contents of AC1 at the time of the termination. This offset is valid only on a trap (?TMXTC contains ?XTR16 or ?XTR32). The high-order word is undefined if a 16-bit process terminated.
?TMAC2 (double-word)	The operating system returns the contents of AC2 at the time of the termination. This offset is valid only on a trap (?TMXTC contains ?XTR16 or ?XTR32). The high-order word is undefined if a 16-bit process terminated.
?TMAC3 (double-word)	The operating system returns the contents of AC3 at the time of the termination. This offset is valid only on a trap (?TMXTC contains ?XTR16 or ?XTR32). The high-order word is undefined if a 16-bit process terminated.

(continued)

Table 2–63.2. Contents of Termination Message from a 32–bit B– or C–Type Process

Offset	Contents
?TMCPC (double-word)	The operating system returns the carry bit and program counter contents at the time of the termination. This offset is valid only on a trap (?TMXTC contains ?XTR16 or ?XTR32). The high-order word is undefined if a 16-bit process terminated.
?TMSEC (double-word)	The operating system returns the number of elapsed seconds since the process was created.
?TMCPU (double-word)	The operating system returns the number of milliseconds of CPU time the process used.
?TMBLK (double-word)	The operating system returns the number of blocks read or written.
?TMPGS (double-word)	The operating system returns the page usage over CPU time in pages/second.
?TMPGD (double-word)	The operating system returns the number of page faults since the process was created.
?TMPGF (double-word)	The operating system returns the number of page faults, with no disk I/O, since the process was created.
?TMR4	Reserved. (Set to 0.)
?TMR5	Reserved. (Set to 0.)
?TMR6	Reserved. (Set to 0.)
?TMR7	Reserved. (Set to 0.)
?TMR8	Reserved. (Set to 0.)
?TMR9	Reserved. (Set to 0.)
?TMTCD	The operating system returns the trap code.
?TMMLG	The operating system returns the number of words in any user-supplied message. This number might be zero.
?TMMSG	Beginning of the message area, starting with two words; ending with the zero or more words in any user-supplied message.

(concluded)

	0 15
?TMXTC	Extended termination code
?TMPRV	Packet revision number
?TMRS	Reserved (Set to 0.)
?TMPID	PID of terminated process
?TMUPD	Reserved (Set to 0.)
?TMUPD + 1	Reserved (Set to 0.)
.	.
.	.
.	.
?TMUPD + 7	Reserved (Set to 0.)
?TMR2	Reserved (Set to 0.)
?TMR3	Reserved (Set to 0.)
?TM0H	AC0 of terminated process (high order)
?TM0L	AC0 of terminated process (low order)
?TM1H	AC1 of terminated process (high order)
?TM1L	AC1 of terminated process (low order)
?TM2H	AC2 of terminated process (high order)
?TM2L	AC2 of terminated process (low order)
?TM3H	AC3 of terminated process (high order)
?TM3L	AC3 of terminated process (low order)
?TMLH	C | Program Counter (high order)
?TMLL	C | Program Counter (low order)
?TMSH	Elapsed seconds since process creation (high order)
?TMSL	Elapsed seconds since process creation (low order)
?TMCH	Milliseconds of CPU time used(high order)
?TMCL	Milliseconds of CPU time used (low order)

(continued)

Figure 2–82.2. Structure of Termination Message from a 16–bit B– or C–Type Process

	0 15
?TMBH	Number of blocks read or written (high order)
?TMBL	Number of blocks read or written (low order)
?TMGH	Page-milliseconds used (high order)
?TMGL	Page-milliseconds used (low order)
?TMDH	Page faults since creation (high order)
?TMDL	Page faults since creation (low order)
?TMFH	Page faults -- no disk I/O (high order)
?TMFL	Page faults -- no disk I/O (low order)
?TMR4	Reserved (Set to 0.)
?TMR5	Reserved (Set to 0.)
?TMR6	Reserved (Set to 0.)
?TMR7	Reserved (Set to 0.)
?TMR8	Reserved (Set to 0.)
?TMR9	Reserved (Set to 0.)
?TMTCD	Trap code
?TMMLG	Message length
?TMMSG	First word of default message
?TMMSG + 1	Second word of default message
?TMMSG + 2	First word of user-supplied message
?TMMSG + 3	Second word of user-supplied message
.	.
.	.
.	.
?TMMSG + (n-2)	Next to last word of user-supplied message
?TMMSG + (n-1)	Last word of user-supplied message

?TDFL = packet length for no user supplied message

(concluded)

Figure 2–82.2. Structure of Termination Message from a 16–bit B– or C–Type Process

Table 2–64. Contents of Termination Message from a 16–bit B– or C–Type Process

Offset	Contents
?TMXTC	The operating system returns the extended termination code to indicate the type of process termination. The values and meanings of these codes are as follows. ?XT16T -- a 16-bit termination of self occurred. ?XTR16 -- a 16-bit user trap occurred. ?XTCIN -- termination by a terminal interrupt. ?XTSUP -- termination by a superior process. ?XTAOS -- termination by AOS. ?XTBCX -- customer connection broken. ?XTCCX -- customer chained. ?XTABR -- customer did ?TABT. ?XT32T -- a 32-bit termination of self occurred. ?XTR32 -- a 32-bit user trap occurred.
?TMPRV	Packet revision number. Place ?TM6 here.
?TMRS	Reserved. (Set to 0.)
?TMPID	The operating system returns the PID of the terminated process.
?TMUPD (8 words)	Reserved. (Set to 0.)
?TMR2	Reserved. (Set to 0.)
?TMR3	Reserved. (Set to 0.)
?TM0H	The operating system returns the high-order contents of AC0 at the time of the termination. This offset is valid only on a trap (?TMXTC contains ?XTR16 or ?XTR32). This offset is undefined if a 16-bit process terminated.
?TM0L	The operating system returns the low-order contents of AC0 at the time of the termination. This offset is valid only on a trap (?TMXTC contains ?XTR16 or ?XTR32).
?TM1H	The operating system returns the high-order contents of AC1 at the time of the termination. This offset is valid only on a trap (?TMXTC contains ?XTR16 or ?XTR32). This offset is undefined if a 16-bit process terminated.
?TM1L	The operating system returns the low-order contents of AC1 at the time of the termination. This offset is valid only on a trap (?TMXTC contains ?XTR16 or ?XTR32).
?TM2H	The operating system returns the high-order contents of AC2 at the time of the termination. This offset is valid only on a trap (?TMXTC contains ?XTR16 or ?XTR32). This offset is undefined if a 16-bit process terminated.
?TM2L	The operating system returns the low-order contents of AC2 at the time of the termination. This offset is valid only on a trap (?TMXTC contains ?XTR16 or ?XTR32).

(continued)

Table 2–64. Contents of Termination Message from a 16–bit B– or C–Type Process

Offset	Contents
?TM3H	The operating system returns the high-order contents of AC3 at the time of the termination. This offset is valid only on a trap (?TMXTC contains ?XTR16 or ?XTR32). This offset is undefined if a 16-bit process terminated.
?TM3L	The operating system returns the low-order contents of AC3 at the time of the termination. This offset is valid only on a trap (?TMXTC contains ?XTR16 or ?XTR32).
?TMLH	The operating system returns the carry bit and high-order program counter contents if and when a 32-bit process terminated. This offset is valid only on a trap (?TMXTC contains ?XTR16 or ?XTR32). The offset is undefined if a 16-bit process terminated.
?TMLL	The operating system returns the low-order program counter contents if and when a 32-bit process terminated, or else the carry bit and program counter contents if and when a 16-bit process terminated. The offset is valid only on a trap (?TMXTC contains ?XTR16 or ?XTR32).
?TMSH	The operating system returns the number of elapsed seconds since the process was created (high order).
?TMSL	The operating system returns the number of elapsed seconds since the process was created (low order).
?TMCH	The operating system returns the number of milliseconds of CPU time the process used (high order).
?TMCL	The operating system returns the number of milliseconds of CPU time the process used (low order).
?TMBH	The operating system returns the number of blocks read or written (high order).
?TMBL	The operating system returns the number of blocks read or written (low order).
?TMGH	The operating system returns the page usage over CPU time in pages/second (high order).
?TMGL	The operating system returns the page usage over CPU time in pages/second (low order).
?TMDH	The operating system returns the number of page faults since the process was created (high order).
?TMDL	The operating system returns the number of page faults since the process was created (low order).

(continued)

Table 2–64. Contents of Termination Message from a 16–bit B– or C–Type Process

Offset	Contents
?TMFH	The operating system returns the number of page faults, with no disk I/O, since the process was created (high order).
?TMFL	The operating system returns the number of page faults, with no disk I/O, since the process was created (low order).
?TMR4	Reserved. (Set to 0.)
?TMR5	Reserved. (Set to 0.)
?TMR6	Reserved. (Set to 0.)
?TMR7	Reserved. (Set to 0.)
?TMR8	Reserved. (Set to 0.)
?TMR9	Reserved. (Set to 0.)
?TMTCD	The operating system returns the trap code.
?TMMLG	The operating system returns the number of words in any user-supplied message. This number might be zero.
?TMMSG	Beginning of the message area, starting with two words; ending with the zero or more words in any user-supplied message.

(concluded)

Notes

- See the descriptions of ?ILKUP, ?TERM, ?RETURN, and ?ISEND in this chapter.

?IRMV
error return
normal return

Input

AC0	Device code of the user device that you want to remove
AC1	Reserved (Set to 0.)
AC2	Reserved (Set to 0.)

Output

AC0	Unchanged
AC1	Undefined
AC2	Undefined

Error Codes in AC0

ERDNM	Illegal device code
ERPRE	Invalid system call parameter
ERPRV	Caller not privileged for this action
ERPTY	Illegal process type

Why Use It?

?IRMV revokes a previous ?IDEF; that is, it allows you to remove a device that you defined earlier in your program. ?IRMV removes only a specific user–defined device.

Who Can Use It?

To issue ?IRMV, a process must have privilege ?PVDV. There are no restrictions concerning file access.

What It Does

?IRMV removes a device's DCT entry from the interrupt vector table. After the operating system executes ?IRMV, it ignores the target device and all subsequent interrupts from it.

Before you issue ?IRMV, load AC0 with the device code you specified when you defined the device with ?IDEF.

Notes

- See the description of ?IDEF in this chapter.

?ISEND *[header address]*
error return
normal return

Input

AC0	Reserved (Set to 0.)
AC1	Reserved (Set to 0.)
AC2	Address of the ?ISEND header, unless you specify the address as an argument to ?ISEND

Output

AC0	Undefined
AC1	Undefined
AC2	Address of the ?ISEND header

Error Codes in AC0

ERIDP	Illegal destination port
ERPRV	Caller not privileged for this action
ERVWP	Invalid word pointer passed as a system call argument
ERSPC	File space exhausted (The IPC spool file is full.)

Why Use It?

?ISEND allows you to pass freeform messages from your current process to a specific receiving process. The ?ISEND and ?IREC system calls are useful for passing variables and other data from one process to another and for synchronizing two processes.

Who Can Use It?

To issue ?ISEND, a process must have privilege ?PVIP or be sending a message to a customer. There are no restrictions concerning file access.

What It Does

?ISEND opens an IPC port for the calling process and sends the message that you specify in the ?ISEND header to a designated receiver.

Before you issue ?ISEND, define the message and the ?ISEND header in your logical address space. You can either cite the header address as an argument to ?ISEND or you can load the address into AC2 before you issue ?ISEND. The ?ISEND header consists of ?IPLTH words. Figure 2–83 shows the ?ISEND header's structure and Table 2–65 describes its contents.

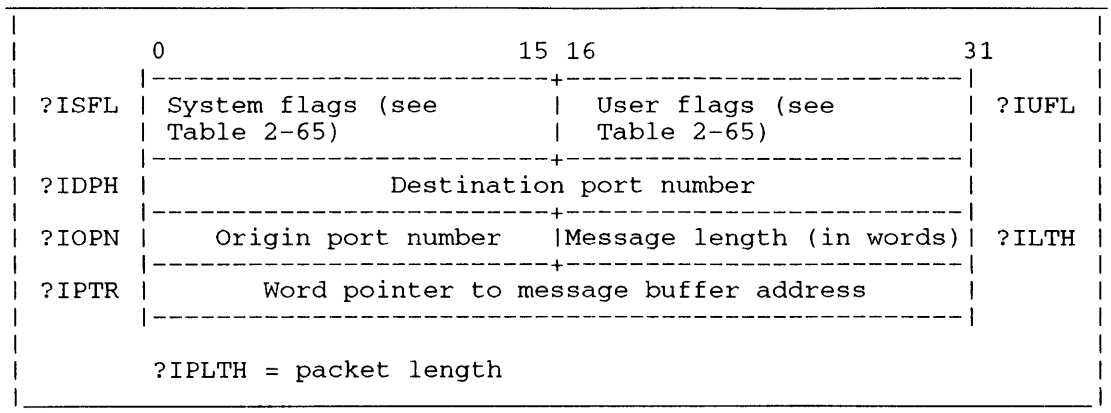


Figure 2–83. Structure of ?ISEND Header

Table 2–65. Contents of ?ISEND Header*

Offset	Contents
?ISFL	System flag word ?IFSTM--Loop the message. (Send the message back to the sender.) ?IFNSP--Do not spool the message; signal an error if there is no ready receiver.
?IUFL	User flag word. (See Figure 2-82.)
?IDPH (doubleword)	Destination port number.
?IOPN	Origin port number.
?ILTH	Message buffer length, in words.
?IPTR (doubleword)	Address of the message buffer.

* There is no default unless otherwise specified.

Offset ?IDPH specifies the destination's global port number. (The ?ILKUP and ?TPORT system calls return this information.) Set offset ?IOPN (the origin port) to the local port number you want to use for this ?ISEND.

Set offset ?IPTR to the address of the message in your logical address space. Specify the length of the message buffer (that is, the length of the message) in offset ?ILTH. If ?ILTH is zero, the operating system sends the message and does not verify ?IPTR. If ?ILTH is nonzero, the operating system sends the message and also verifies ?IPTR.

Sample Header

```
HDR:    .BLK    ?IPLTH          ;Header length.
        .LOC    HDR+?ISFL       ;System flags.
        .WORD   0               ;Set to 0.
        .LOC    HDR+?IUFL       ;User flags.
        .WORD   0               ;Set to 0.
        .LOC    HDR+?IDPH       ;Global port number of receiving
        .DWORD  5               ;process.  (?ILKUP and ?TPORT return
                                ;this information.)
        .LOC    HDR+?IOPN       ;Origin port no.
        .WORD   7               ;Local port no. to use for this
                                ;?ISEND.
        .LOC    HDR+?ILTH       ;Message buffer length (words).
        .WORD   7               ;Set buffer length before each
                                ;?ISEND.
        .LOC    HDR+?IPTR       ;Message buffer address (must be in
        .DWORD  SBUFF           ;unshared area of your logical
                                ;address space).
        .LOC    HDR+?IPLTH      ;End of ?ISEND header.
```

Notes

- See the descriptions of ?IREC, ?IS.R, ?ILKUP, and ?TPORT in this chapter.
- In the explanation of ?PROC, see the section "Offset ?PIPC." It explains how ?PROC can send a CLI–format command line as the IPC message.
- A global port number contains the PID of the target process. In order to ensure the global port number remains unique, a connection must exist between the calling and the target processes. This connnection will cause the operating system to reserve the PID portion of the global port until the connection is explicitly broken. If an ?ISEND is attempted when the process described by the global port number has terminated, connection management will guarantee the ERIDP error will be returned until the calling process does the ?DCON. See the description of ?CON, ?DCON, ?DRCON, ?RESIGN?, and ?SERVE in this chapter.

?ISPLIT
error return
normal return

Input

AC0	One of the following: • Global port number (32–bit users only) • OPH, which indicates high–order bits of global port number (16–bit users only)
AC1	One of the following: • Reserved (Set to 0.) (32–bit users only) • OPL, which indicates low–order bits of global port number (16–bit users only)
AC2	Reserved (Set to 0.)

Output

AC0	One of the following: • Ring field in Bits 29 through 31 (32–bit users only) • Ring field in Bits 13 through 15 (16–bit users only)
AC1	PID of the process that owns the port
AC2	Local port number

Error Codes in AC0

No error codes are currently defined.

Why Use It?

?ISPLIT, like ?IMERGE, allows both 16– and 32–bit users to manipulate ring fields within global port numbers.

Who Can Use It?

There are no special process privileges needed to issue this call, and there are no restrictions concerning file access.

What It Does

?ISPLIT returns the ring field, the PID, and the local port number associated with the global port number you specify in AC0.

Notes

- See the description of ?IMERGE in this chapter.

?IS.R *[packet address]*
error return
normal return

Input

AC0	Reserved (Set to 0.)
AC1	Reserved (Set to 0.)
AC2	Address of the IS.R header, unless you specify the address as an argument to ?IS.R

Output

AC0	Undefined
AC1	Undefined
AC2	Address of the ?IS.R header

Error Codes in AC0

ERIDP	Illegal destination port
ERMPR	System call parameter address error (The receive buffer is not in the unshared area of your address space.)
ERPRV	Caller not privileged for this action
ERVWP	Invalid word pointer passed as a system call argument

Why Use It?

?IS.R allows you to send a message to a receiver and wait for a reply on the same port. ?IS.R performs a special case of ?ISEND followed by an ?IREC with a slightly lower overhead.

Who Can Use It?

To issue ?IS.R, a process must have privilege ?PVIP or have a connection to the target process. There are no restrictions concerning file access.

What It Does

?IS.R performs an ?ISEND and, if successful, follows it immediately with an ?IREC.

Before you issue ?IS.R, you must define the message and set up a receive buffer (for the receiver's reply) in your logical address space. In addition, you must set up the ?IS.R packet, a combination of the ?ISEND and ?IREC packets that consists of ?IPRLTH words. You can cite the packet address as an argument to ?IS.R or load its address into AC2 before you issue ?IS.R. Figure 2–84 shows the structure of the ?IS.R packet, and Table 2–66 describes its contents.

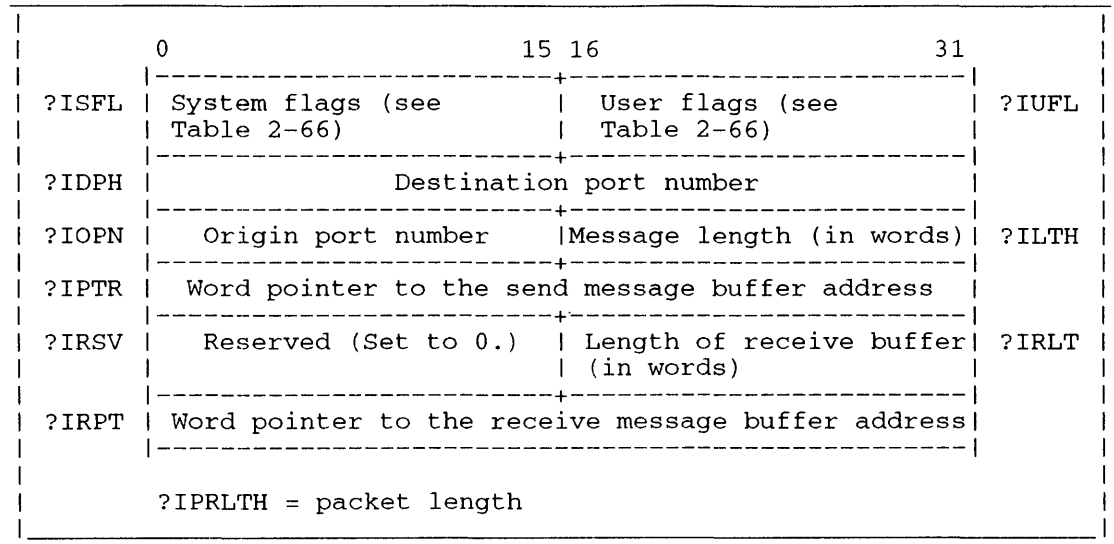


Figure 2–84. Structure of ?IS.R Header

The message is sent and the reply is received, but not necessarily with the same port numbers. (The ?TPORT system call translates a local port number to its global equivalent.)

During message transmission, the operating system suspends the ?IS.R calling task until it receives the reply or until it takes an error return.

Table 2–66. Contents of ?IS.R Header*

Offset	Contents
?ISFL	System flag word ?IFSTM--Loop the message. (Send the message back to the sender.) ?IFNSP--Do not spool the message; signal an error if there is no ready receiver. ?IFRFM--Receive a looped message (sent by this process to itself). ?IFSOV--Spool the message if the receive buffer is too small. ?IFNBK--Signal an error if there is no spooled message for this receiver. ?IFRING--Contains the sender's ring field (returned by the OS). (This is a 3-bit quantity.) ?IFPR---Indicates .PR file type of sender; 0 if sender is a 32-bit process; 1 if sender is a 16-bit process (returned by the OS).
?IUFL	User flag word. (See Figure 2-82.)
?IDPH (doubleword)	Destination port number.
?IOPN	Origin port number.
?ILTH	Length of the message buffer, in words.
?IPTR (doubleword)	Word pointer to the address of the message buffer.
?IRSV	Reserved. (Set to 0.)
?IRLT	Length of the receive buffer, in words.
?IRPT (doubleword)	Word pointer to the address of the receive buffer.

* There is no default unless otherwise specified.

Sample Header

```
HDR:    .BLK    ?IPRLTH         ;Allocate enough space for the packet
                                ;Packet length = ?IPRLTH.
        .LOC    HDR+?ISFL       ;System flags.
        .WORD   0               ;None.
        .LOC    HDR+?IUFL       ;User flags.
        .WORD   0               ;Set to 0.
        .LOC    HDR+?IDPH       ;Global port no. of receiving process.
        .DWORD  5               ;(?TPORT returns this information.)
        .LOC    HDR+?IOPN       ;Origin port no.
        .WORD   7               ;Local port no. to use for this ?IS.R.
        .LOC    HDR+?ILTH       ;Message buffer length (in words).
        .WORD   0               ;Set buffer length before each ?IS.R.
        .LOC    HDR+?IPTR       ;Message buffer address (must be in
        .DWORD  BUFF*2          ;unshared area of your logical address
                                ;space).  Byte pointer to BUFF is
                                ;message buffer address.
        .LOC    HDR+?IRSV       ;Reserved
        .WORD   0               ;You must set this value to 0.
        .LOC    HDR+?IRLT       ;Receive buffer length (in words).
        .WORD   0
        .LOC    HDR+?IRPT       ;Receive buffer address.
        .DWORD  -1
        .LOC    HDR+?IPLTH      ;End of ?ISEND header.
```

Notes

- See the descriptions of ?ISEND, ?IREC, and ?TPORT in this chapter.

?ITIME
error return
normal return

Input

None

Output

AC0 Undefined

AC1 Time since midnight in 32,768ths of a second

AC2 Days since 1 January 1968; on 1 January 1968 AC2 contained 1, not 0

Error Codes in AC0

No error codes are currently defined.

Why Use It?

?ITIME provides you with a time stamp.

Who Can Use It?

There are no special process privileges needed to issue this call, and there are no restrictions concerning file access.

What It Does

?ITIME returns the current date and time in AC1 and AC2. However, although ?ITIME returns the time to 32,768ths of a second, it is only as accurate as the real–time clock. To obtain the frequency — and therefore the accuracy — of the real–time clock, issue the ?GHRZ system call.

```
?IXIT
                    ; no error return
                    ; no normal return
```

Input		Output	
AC0	Reserved (Set to 0.)	AC0	Undefined
AC1	One of the following: • 0 to suppress task scheduling upon return • Any nonzero value, to re–enable task scheduling	AC1	Unchanged
AC2	Reserved (Set to 0.)	AC2	Undefined

Error Codes in AC0

None (There is no error return.)

Why Use It?

You must issue ?IXIT if you want to return the calling process from an interrupt service routine. Once a process has entered such a routine, FIXMT, ?IXIT, ?IXMT (which transmits an interrupt message) and ?SIGNL are the only system calls that process can issue. You cannot issue these system calls from any other kinds of routines.

Who Can Use It?

There are no special process privileges needed to issue this call, and there are no restrictions concerning file access.

What It Does

?IXIT returns the calling process from an interrupt service routine. (Control passes to the operating system after this system call.) By loading AC1 with 0, you can direct the operating system to suppress task scheduling upon return from the interrupt service routine.

Before issuing ?IXIT you must restore the contents of all the stack registers (WFP, WSB, WSL, WSP). These registers must contain the same values they had when you entered your interrupt service routine. Not restoring these contents will likely cause a panic after a few interrupts.

Notes

- See the description of ?IXMT in this chapter.

?IXMT Transmits a message from an interrupt service routine.

?IXMT
error return
normal return

Input		Output	
AC0	Reserved (Set to 0.)	AC0	Undefined
AC1	Message	AC1	Unchanged
AC2	DCT address of the sending device	AC2	Unchanged

Error Codes in AC0

ERXMT	Signal to address already in use (that is, the mailbox)
ERXMZ	Attempt to XMT illegal message

Why Use It?

?IXMT and ?IMSG allow you to send data from an interrupt service routine to an outside task, or to synchronize the routine with an outside task.

Who Can Use It?

There are no special process privileges needed to issue this call beyond those that ?IDEF requires, and there are no restrictions concerning file access.

What It Does

?IXMT sends a message up to 32 bits long from an interrupt service routine to a specific receiving task outside the sending routine. (The receiving task issues ?IMSG to receive the message.) ?FIXMT, ?IXIT, ?IXMT, and ?SIGNL are the only system calls you can issue from an interrupt service routine. In addition, you cannot issue system calls ?IXIT and ?IXMT from any other kind of routine.

Before you issue ?IXMT, load AC2 with the user DCT address of the device associated with the sending routine, and load AC1 with the message. The message must be nonzero, or ?IXMT fails on error ERXMZ.

When the operating system executes ?IXMT, it passes the message to the mailbox that it associates with the device. (Be sure to initialize the mailbox before you issue ?IXMT.) Then, when the receiving task issues the complementary ?IMSG, the operating system passes the message from the mailbox to AC1. If the mailbox already contains a nonzero value, the ?IXMT fails on error code ERXMT.

?IXMT Continued

If a sending routine issues ?IXMT before the receiver issues ?IMSG, the operating system holds the message in the mailbox for later delivery. If the ?IMSG occurs before the ?IXMT, the operating system suspends the receiving task until the transmission occurs.

After returning from ?IXMT, AC3 contains the return address from ?IXMT. This is different from almost all other system calls, which place the contents of the frame pointer in AC3.

Notes

- See the descriptions of ?IDEF, ?FIXMT, ?IXIT, and ?IMSG in this chapter.

AOS/VS

?JPINIT *[packet address]*
error return
normal return

Input

AC0	Reserved (Set to 0.)
AC1	Reserved (Set to 0.)
AC2	Address of the ?JPINIT packet, unless you specify the address as an argument to ?JPINIT

Output

AC0	Unchanged
AC1	Unchanged
AC2	Address of the ?JPINIT packet

Error Codes in AC0

ERIJP	Invalid JPID
ERILP	LPID does not exist
ERJAI	JP already initialized
ERMFF	Microcode format error
ERNML	No microcode loaded
ERNMP	Not a multiprocessor system
ERPRV	Caller not privileged for this action
ERVBP	Invalid byte pointer passed as system call argument

Why Use It?

Use this system call to initialize a job processor (JP) that is not currently active.

Who Can Use It?

You must be PID 2 or have System Manager privilege to issue this call. There are no restrictions concerning file access.

What It Does

This system call initializes an inactive job processor. AOS/VS attaches the JP to the specified logical processor (LPID in the parameter packet) and begins to schedule processes to run on the JP.

AOS/VS selects microcode for the JP in one of three ways according to the values of two bits in word JPI_PKT.FLAGS of the parameter packet in Figure 2–85.

First, if you set bit ?JPI_PKT.FLAGS_IN_UC (and provide a byte pointer in doubleword ?JPI_PKT.UC_NAME), AOS/VS loads the specified microcode file into the JP. AOS/VS also requires bit ?JPI_PKT.FLAGS_EX_UC to be zero.

Second, if you do not set bit ?JPI_PKT.FLAGS_IN_UC, AOS/VS looks at bit ?JPI_PKT.FLAGS_EX_UC. A value of 1 tells AOS/VS NOT to load microcode into the JP; instead, AOS/VS uses the microcode that is currently in the JP.

Third, if you don't set bits ?JPI_PKT.FLAGS_IN_UC and ?JPI_PKT.FLAGS_EX_UC (i.e., if you don't provide a microcode pathname and don't want to use existing microcode), AOS/VS loads a standard microcode file based on the CPUID of the JP.

Suppose an error occurs while ?JPINIT loads a microcode file. The system console does NOT receive any messages. However, AOS/VS returns an error code to your program.

Figure 2–85 shows the structure of the ?JPINIT parameter packet, and Table 2–67 describes its contents.

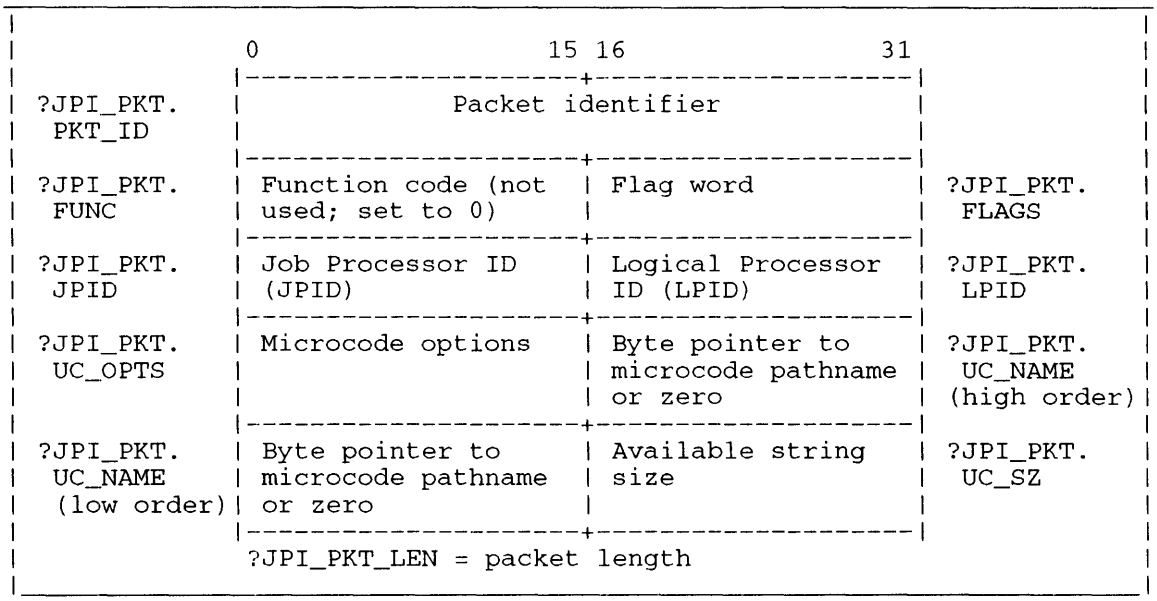


Figure 2–85. Structure of ?JPINIT Packet

Table 2–67. Contents of ?JPINIT Packet

Offset	Contents
?JPI_PKT.PKT_ID (doubleword)	Packet identifier. Place ?JPI_PKT_PKTID here.
?JPI_PKT.FUNC	Function code. Not used. (Set to 0.)
?JPI_PKT.FLAGS	Flag word to control microcode file loading. Legal values of bit pairs ?JPI_PKT.FLAGS_IN_UC and ?JPI_PKT.FLAGS_EX_UC are 10, 01, and 00.
?JPI_PKT.JPID	Job processor ID. It must be between values ?JPID_MIN and ?JPID_MAX, inclusive.
?JPI_PKT.LPID	Logical processor ID. It must be between values ?LPID_MIN and ?LPID_MAX, inclusive.
?JPI_PKT.UC_OPTS	Microcode options word. This offset corresponds directly with bits 0-15 of AC0 for the JPLCS instruction. The options in this offset are specific to the processor and cannot be listed here. However, if this offset contains zero, you have specified all the default options.
?JPI_PKT.UC_NAME (doubleword)	Byte pointer to the microcode file's pathname, or 0 if you are not specifying a pathname.
?JPI_PKT.UC_SZ	Number of bytes in the buffer containing the microcode file's pathname, or 0 if you are not specifying a pathname.

?JPMOV Moves a job processor to a new logical processor.

AOS/VS

?JPMOV *[packet address]*
error return
normal return

Input

AC0	Reserved (Set to 0.)
AC1	Reserved (Set to 0.)
AC2	Address of the ?JPMOV packet, unless you specify the address as an argument to ?JPMOV

Output

AC0	Unchanged
AC1	Unchanged
AC2	Address of the ?JPMOV packet

Error Codes in AC0

ERIJP	Invalid JPID
ERILP	Invalid LPID
ERJNI	JP is not initialized
ERLNE	LP does not exist
ERLJP	Attempt to release last JP attached to an LP
ERJAA	JP already attached to LP
ERPRV	Caller not privileged for this action

Why Use It?

Use this system call to move a job processor (JP) to another logical processor (LP).

Who Can Use It?

You must be PID 2 or have System Manager privilege to issue this call. There are no restrictions concerning file access.

What It Does

This system call moves a job processor, if it is attached to an existing logical processor, to the specified logical processor. After this call executes successfully, AOS/VS schedules jobs on the JP according to the new LP's class designations.

When you move a JP, it's possible that you could leave an LP behind that has no JP attached to it. To prevent this from happening, set a bit in the flag word of the parameter packet. Then, you will get an error if you leave an "unattached" LP. If you do not set this bit, AOS/VS will move the JP regardless of whether or not it would result in an LP without a JP.

Figure 2–86 shows the structure of the ?JPMOV parameter packet, and Table 2–68 describes its contents.

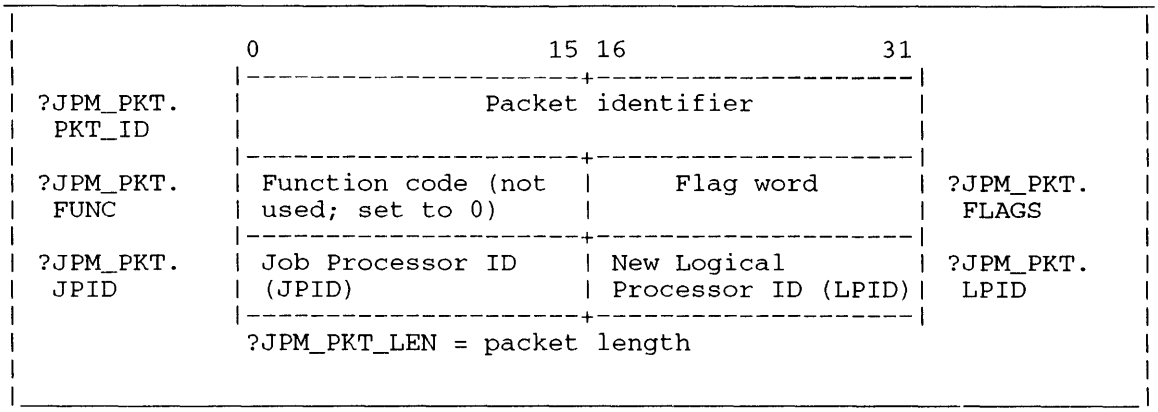


Figure 2–86. Structure of ?JPMOV Packet

Table 2–68. Contents of ?JPMOV Packet

Offset	Contents
?JPM_PKT.PKT_ID (doubleword)	Packet identifier. Place ?JPM_PKT_PKTID here.
?JPM_PKT.FUNC	Function code. Not used. (Set to 0.)
?JPM_PKT.FLAGS	Flag word. If bit ?JPM_PKT.FLAGS_LJPID is set, AOS/VS returns error code ERLJP when you try to move a JP that would leave an unattached LP; the JP remains attached to the LP. If the bit is not set, AOS/VS returns no error code under the same circumstances; the JP is removed from the old LP and attached to the new LP.
?JPM_PKT.JPID	Job processor ID. It must be between values ?JPID_MIN and ?JPID_MAX, respectively.
?JPM_PKT.LPID	New logical processor ID. It must be between values ?LPID_MIN and ?LPID_MAX, respectively.

AOS/VS

?JPREL *[packet address]*
error return
normal return

Input

AC0	Reserved (Set to 0.)
AC1	Reserved (Set to 0.)
AC2	Address of the ?JPREL packet, unless you specify the address as an argument to ?JPREL

Output

AC0	Unchanged
AC1	Unchanged
AC2	Address of the ?JPREL packet

Error Codes in AC0

ERIJP	Invalid JPID
ERJNI	JP is not initialized
ERJST	JP running one or more system tasks
ERLJP	Attempt to release last JP attached to an LP
ERNMP	Not a multiprocessor system
ERPRV	Caller not privileged for this action

Why Use It?

Use this system call to release a job processor (JP) from your AOS/VS system.

Who Can Use It?

You must be PID 2 or have System Manager privilege to issue this call. There are no restrictions concerning file access.

What It Does

This system call releases a job processor.

AOS/VS returns an error if the JP is not initialized. AOS/VS also returns an error if the JP is performing a vital system task such as file I/O.

If the JP is initialized and not performing a vital system task, AOS/VS removes the JP from its associated logical processor (LP) and stops the JP from processing. When you release a JP, it's possible that you could leave an LP behind that has no JP attached to it. To prevent this from happening, set a bit in the flag word of the parameter packet. Then, you will get an error if you leave an "unattached" LP. If you do not set this bit, AOS/VS will remove the JP regardless of whether or not it would result in an LP without a JP.

Figure 2–87 shows the structure of the ?JPREL parameter packet, and Table 2–69 describes its contents.

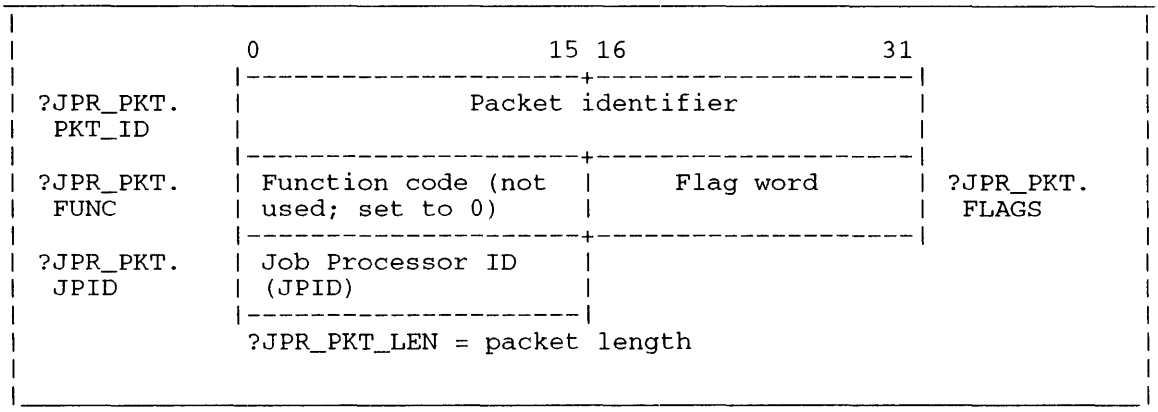


Figure 2–87. Structure of ?JPREL Packet

Table 2–69. Contents of ?JPREL Packet

Offset	Contents
?JPR_PKT.PKT_ID (doubleword)	Packet identifier. Place ?JPR_PKT_PKTID here.
?JPR_PKT.FUNC	Function code. Not used. (Set to 0.)
?JPR_PKT.FLAGS	Flag word. If bit ?JPR_PKT.FLAGS_LJPID is set, AOS/VS returns error code ERLJP when you try to release a JP that would leave an unattached LP; the JP remains attached to the LP. If the bit is not set, AOS/VS returns no error code under the same circumstances; the JP is removed from the LP.
?JPR_PKT.JPID	Job processor ID. It must be between values ?JPID_MIN and ?JPID_MAX, respectively.

?JPSTAT — Gets the status of a job processor.

AOS/VS

?JPSTAT *[packet address]*
error return
normal return

Input

AC0	Reserved (Set to 0.)
AC1	Reserved (Set to 0.)
AC2	Address of the ?JPSTAT packet, unless you specify the address as an argument to ?JPSTAT

Output

AC0	Unchanged
AC1	Unchanged
AC2	Address of the ?JPSTAT packet

Error Codes in AC0

ERIJP	Invalid JPID
ERJNI	JP is not initialized
ERICD	Invalid function code
ERNMP	Not a multiprocessor system

Why Use It?

Use this system call to obtain information about all initialized job processors (JP) or about a specific initialized job processor.

Who Can Use It?

There are no special process privileges needed to issue this call, and there are no restrictions concerning file access.

What It Does

This system call has two versions — general and specific. The general version returns the total number and JPIDs of all initialized job processors. The specific version returns status information of a specific initialized job processor. You tell AOS/VS which version you want by placing a code in offset ?JPS_PKT.FUNC of the main parameter packet.

If you select the general version, then AOS/VS returns, in a doubleword of the general status subpacket, a bit map of initialized job processors that operates as follows:

- If a bit is set, then the JP with the corresponding JPID is initialized.
- If a bit is not set, then the JP with the corresponding JPID is not initialized.

For example, suppose bits 2, 3, and 5 are set. This means JPs with JPIDs of 2, 3, and 5 are initialized. Furthermore, JPs with JPIDs of 0, 1, 4, 6, 7, 8, ..., 15 are not initialized.

- Information that the assembly language instruction JPSTATUS returns. This information is in the five words beginning with ?JPS_SPEC.STATE.
- An indication, in the software flags word, whether the JP is the "mother" processor and whether the JP is running a system task. In either case, you cannot release the JP.
- Which LP the JP is attached to.

If you select the specific version, then AOS/VS returns, in the specific status subpacket, the following information for the target JP:

Figure 2–88 shows the structure of the ?JPSTAT main parameter packet, and Table 2–70 describes its contents. Figure 2–89 and Figure 2–90 show the structure of the general status subpacket and specific status subpacket, respectively. Table 2–71 and Table 2–72 describe the contents of these respective subpackets.

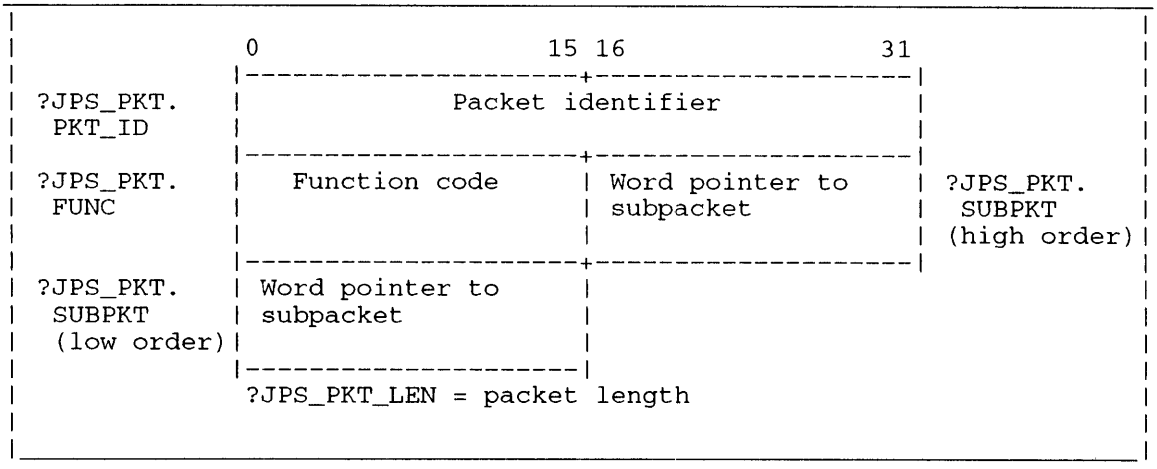


Figure 2–88. Structure of ?JPSTAT Main Packet

Table 2–70. Contents of ?JPSTAT Main Packet

Offset	Contents
?JPS_PKT.PKT_ID (doubleword)	Packet identifier. Place ?JPS_PKT_PKTID here.
?JPS_PKT.FUNC	General/Specific code. Place ?JPS_GEN here to obtain general information or ?JPS_SPEC to obtain specific information.
?JPS_PKT.SUBPKT (doubleword)	If you have selected general information, place the word address of the general status subpacket here. If you have selected specific information, place the word address of the specific status subpacket here.

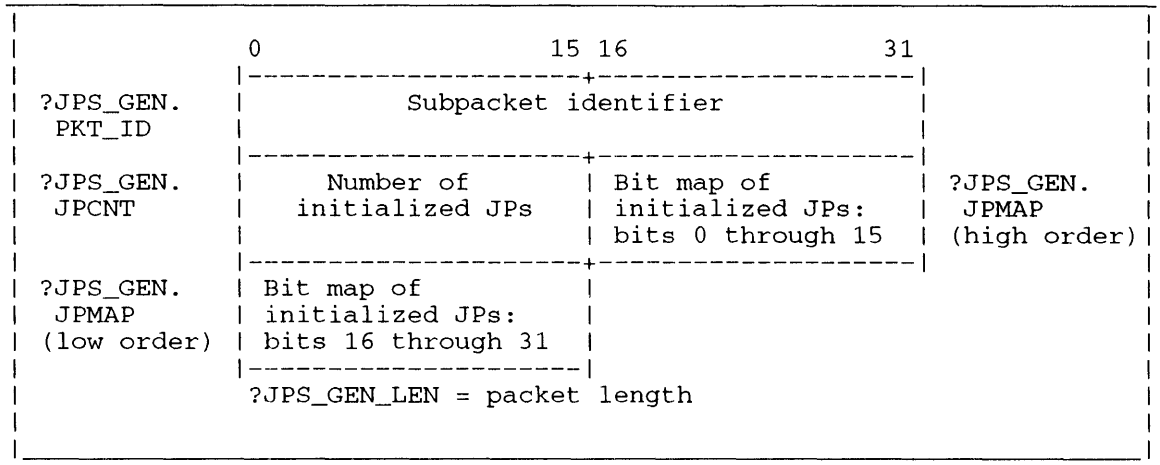


Figure 2–89. Structure of ?JPSTAT General Information Subpacket

Table 2–71. Contents of ?JPSTAT General Information Subpacket

Offset	Contents
?JPS_GEN.PKT_ID (doubleword)	Subpacket identifier. Place ?JPS_GEN_PKTID here.
?JPS_GEN.JPCNT	Number of job processors that are currently initialized.
?JPS_GEN.JPMAP (doubleword)	Bit map to indicate the JPIDs of the job processors that are currently initialized. For example, if AOS/VS has set the leftmost bit of the first (high order) word in this double word, then job processor number 0 is currently initialized.

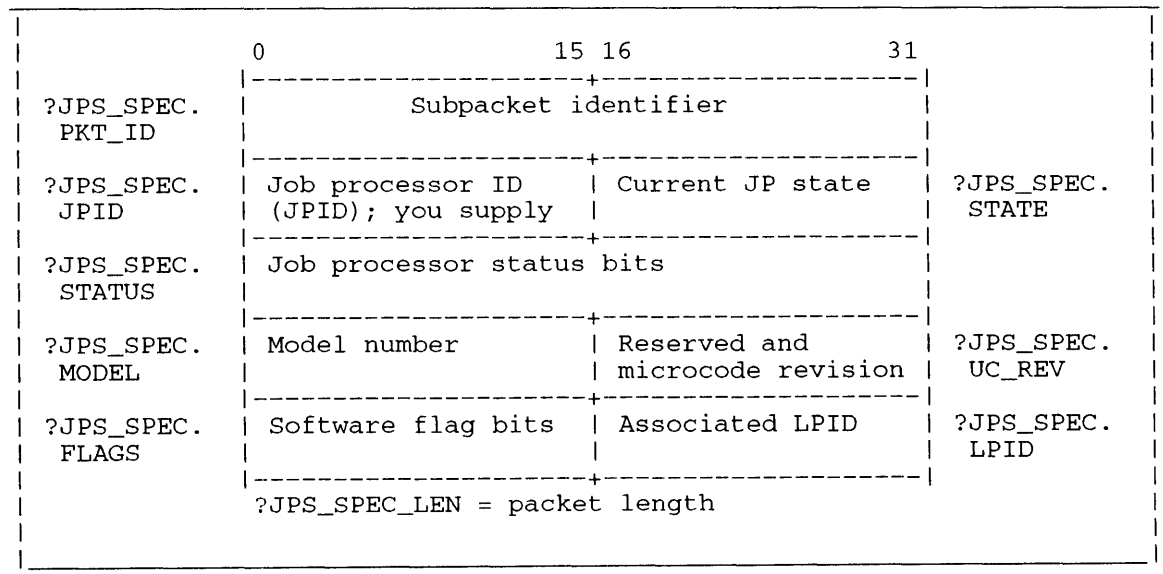


Figure 2–90. Structure of ?JPSTAT Specific Information Subpacket

Table 2–72. Contents of ?JPSTAT Specific Information Subpacket

Offset	Contents
?JPS_SPEC.PKT_ID (doubleword)	Subpacket identifier. Place ?JPS_SPEC_PKTID here.
?JPS_SPEC.JPID	ID number of the job processor that you want to receive information about.
?JPS_SPEC.STATE	Current state of the JP.
?JPS_SPEC.STATUS (doubleword)	JP status bits.
?JPS_SPEC.MODEL	JP model number.
?JPS_SPEC.UC_REV	Left byte is reserved; right byte contains the microcode revision number.
?JPS_SPEC.FLAGS	Software flag bits. If bit ?JPS_SPEC.FLAGS_MOM is set, then the JP is the mother processor. If bit ?JPS_SPEC.FLAGS_SYSTSK is set, then the JP is running a system task.
?JPS_SPEC.LPID	ID number of the logical processor that is associated with the job processor you specified in offset ?JPS_SPEC.JPID.

?KCALL — Keeps the calling resource and acquires a new resource (16–bit processes only).

?KCALL *[procedure entry]*
normal return
alternate return

Input

The OS passes the input accumulators and the carry bit to the new procedure

Output

The accumulators and the carry bit are unchanged, unless they are modified by the new procedure

Error Codes in AC0

The following error codes can be passed to the ?BOMB routine:

ERICM	Illegal system command
ERLRF	Overlay load error
ERSEN	Invalid shared library reference (The OS does not support shared libraries.)

Why Use It?

In general, you issue ?KCALL instead of ?RCALL only when you want to avoid releasing the calling resource; for example, to guarantee the correct return address within a movable resource.

Who Can Use It?

There are no special process privileges needed to issue this call, and there are no restrictions concerning file access.

What It Does

?KCALL keeps the calling resource, loads a new resource, and transfers control to the specified procedure entry in the new resource. You can pass the procedure entry as an argument to ?KCALL or pass it on the stack (via the .PTARG pseudo–op). If you pass the procedure entry on the stack, the operating system pops it off as it executes ?KCALL.

After the operating system executes the target procedure, it returns control to the normal return, unless the target procedure specified an alternate return by issuing an ISZ ?ORTN,3 instruction followed by an RTN instruction.

If the calling task terminates, the operating system releases the newly acquired resource, but does not release the calling task. Therefore, do not terminate a task until all its outstanding ?KCALLs are completed.

Because ?KCALL does not release the calling resource, it requires more memory than ?RCALL. ?KCALL can also cause resource deadlocks. We recommend that you use ?RCALL instead of ?KCALL in most cases.

Notes

- See the description of ?RCALL in this chapter.

?KHIST — Kills a histogram.

?KHIST
error return
normal return

Input

None

Output

None

Error Codes in AC0

ERHIS Error on histogram init/delete (caller has no active histogram)

Why Use It?

?KHIST explicitly stops histogram monitoring of a process. If you do not issue ?KHIST, the histogram continues until the process that initiated it terminates.

Who Can Use It?

There are no special process privileges needed to issue this call, and there are no restrictions concerning file access.

What It Does

?KHIST stops the histogram started by a previous ?WHIST or ?IHIST system call. ?KHIST requires no input parameter, because a process can run only one histogram at a time.

Notes

- See the description of ?IHIST and ?WHIST in this chapter.

?KILAD
error return
normal return

Input		Output	
AC0	Address of the kill–processing routine	AC0	Unchanged
AC1	Reserved (Set to 0.)	AC1	Undefined
AC2	Reserved (Set to 0.)	AC2	Undefined

Error Codes in AC0

ERVWP	Invalid word pointer passed as a system call argument

Why Use It?

In general, you define a kill–processing routine to release a task's resources in an orderly fashion when it terminates. For example, a kill–processing routine might consist of stack release instructions and explicit instructions to release any channels or user devices that are currently open. You should end a kill–processing routine with ?KILL to terminate the calling task.

?KILAD defines a kill–processing routine for the calling task only. You can also set up a generalized kill–processing routine, ?UKIL, for all tasks in a process. You can execute both ?KILAD and ?UKIL processing for any task.

Who Can Use It?

There are no special process privileges needed to issue this call, and there are no restrictions concerning file access.

What It Does

?KILAD defines a kill–processing routine for the calling task. The kill–processing routine gains control the first time another task issues ?IDKIL or ?PRKIL to terminate the ?KILAD caller. (The operating system ignores the kill–processing routine when a task terminates with ?KILL or with a WRTN instruction.)

Before you issue ?KILAD, load AC0 with the address of the kill–processing routine (defined elsewhere in your program). When a task enters a kill–processing routine, the operating system assigns it the highest priority level (0) so that it will gain control as soon as possible. (It may gain control immediately, if there are no other tasks at priority level 0.)

Notes

- See the descriptions of ?IDKIL, ?PRKIL, and ?KILL in this chapter.

?KILL
error return

Input

None

Output

None

Error Codes in AC0

ERLTK Last task was killed

Why Use It?

?KILL terminates the task that is currently executing. ?KILL also moves the terminated task's system databases to the inactive queue for eventual use in initiating another task.

Who Can Use It?

There are no special process privileges needed to issue this call, and there are no restrictions concerning file access.

What It Does

?KILL invokes the ?UKIL task–termination routine (either the user–defined ?UKIL or the default), and terminates the calling task unconditionally.

When you use ?KILL to kill the last active, ready task in the process, the operating system returns error code ERLTK to AC0. You must end the last executing task in a process by issuing ?RETURN.

You can use ?KILL to terminate only the calling task. (The ?IDKIL and ?PRKIL system calls allow you to terminate the calling task or any other task that you specify by TID and priority, respectively.)

Notes

- See the descriptions of ?KILAD, ?IDKIL, ?PRKIL, and. ?RETURN in this chapter.

?KINTR
error return
normal return

Input

AC0	Second character of a two–control–character, terminal–interrupt sequence in low order byte
AC1	PID to interrupt (must be a customer)
AC2	Unused

Output

AC0	Unchanged
AC1	Unchanged
AC2	Unchanged

Error Codes

ERPRH	Process is not in hierarchy (The PID that you tried to interrupt is not a customer.)

Why Use It?

?KINTR performs remote terminal interrupts. This allows virtual terminals to behave as if they were real terminals.

Who Can Use It?

There are no special process privileges needed to issue this call (although you can interrupt only a customer), and there are no restrictions concerning file access.

What It Does

?KINTR simulates keyboard interrupt sequences between processes that share a customer/server connection when the customer is waiting because of a ?KWAIT system call.

Notes

- See the description of ?KWAIT in this chapter.
- See the description of ?OBEL, ?INTWT, and ?ODIS in this chapter. These system calls handle CRTL–C CTRL–A console interrupts only.

?KIOFF Disables control–character terminal interrupts.

?KIOFF
error return
normal return

Input

AC0	Unused
AC1	Unused
AC2	Unused

Output

AC0	Unchanged
AC1	Unchanged
AC2	Unchanged

Error Codes

None

Why Use It?

?KIOFF prevents a process from being interrupted by a terminal control sequence.

Who Can Use It?

There are no special process privileges needed to issue this call, and there are no restrictions concerning file access.

What It Does

?KIOFF allows you to disable all two–control–character, terminal–interrupt sequences.

Notes

- See the description of ?KION in this chapter.

?KION
error return
normal return

Input		Output	
AC0	Unused	AC0	Unchanged
AC1	Unused	AC1	Unchanged
AC2	Unused	AC2	Unchanged

Error Codes

No error codes are currently defined.

Why Use It?

?KION re–enables terminal interrupts that were previously disabled by the ?KIOFF system call.

Who Can Use It?

There are no special process privileges needed to issue this call, and there are no restrictions concerning file access.

What It Does

?KION allows you to re–enable all two–control–character, terminal–interrupt sequences.

Notes

- See the description of ?KIOFF in this chapter.

?KWAIT
error return
normal return

Input

AC0	Unused
AC1	Unused
AC2	Unused

Output

AC0	Second character of a two–control–character, terminal–interrupt sequence in low–order byte
AC1	Unchanged
AC2	Unchanged

Error Codes

ERKAD Interrupt task already defined

Why Use It?

?KWAIT allows a process to handle terminal interrupts according to your programming requirements.

Who Can Use It?

There are no special process privileges needed to issue this call, and there are no restrictions concerning file access.

What It Does

?KWAIT allows a process or task to wait for a terminal interrupt. When the interrupt occurs, the operating system passes the second character of the terminal control sequence to the waiting process. The waiting process can then take whatever action the control character directed.

When the terminal is a shared console (characteristic ?SHC0 of ?GCHR system call), the control sequences are deflected back to the owner of the console.

Notes

- See the description of ?KINTR in this chapter.
- See the description of ?OBEL, ?ODIS, and ?INTWT in this chapter. These system calls handle CTRL–C CTRL–A console interrupts only.

?LABEL — Creates a label for a magnetic tape or diskette.

AOS/VS

?LABEL *[packet address]*
error return
normal return

Input

AC0	Reserved (Set to 0.)
AC1	Reserved (Set to 0.)
AC2	Address of the ?LABEL packet, unless you specify the address as an argument to ?LABEL

Output

AC0	Undefined
AC1	Undefined
AC2	Address of the ?LABEL packet

Error Codes in AC0

ERBDK	Bad diskette
ERCDN	Controller does not support this density mode
EREO1	File is open, can't exclusively open
EREO2	File is exclusively opened, can't open
ERFAD	File access denied
ERIDT	Illegal device name type
ERILF	Illegal label format
ERILL	Illegal labeled tape level
ERMPR	System call parameter address error
EROTL	Owner ID too long
ERPRE	Invalid system call parameter
ERPUF	Physical unit failure
ERUTL	User labels too long or too many
ERVOL	Incorrect volume mounted
ERVTL	Volid too long or null
ERVWP	Invalid word pointer passed as a system call argument

Why Use It?

?LABEL allows you to label a magnetic tape volume. (Another way to do this is to use the CLI LABEL utility.) ?LABEL also allows you to label a diskette.

Who Can Use It?

There are no special process privileges needed to issue this call. You must have Write access to the magnetic tape or diskette unit.

What It Does

?LABEL allows you to label a magnetic tape or diskette under program control. To use ?LABEL, you must supply a packet ?LBLN words long. You can specify the packet address in AC2 before you issue ?LABEL or you can use the packet address as an argument. Figure 2–91 shows the structure and Table 2–73 describes the contents of the ?LABEL packet.

Offset ?LBFG contains status bits that allow you to specify the tape unit's density mode and whether you want the label in IBM format. There are three density mode settings (comparable to those in the ?OPEN packet): ?LB8 sets the tape unit to a density of 800 bytes/inch; ?LB16 sets the unit to 1600 bytes/inch; ?LB62 sets the tape unit to a density of 6,250 bytes/inch; ?LBAM directs the operating system to set the correct density automatically.

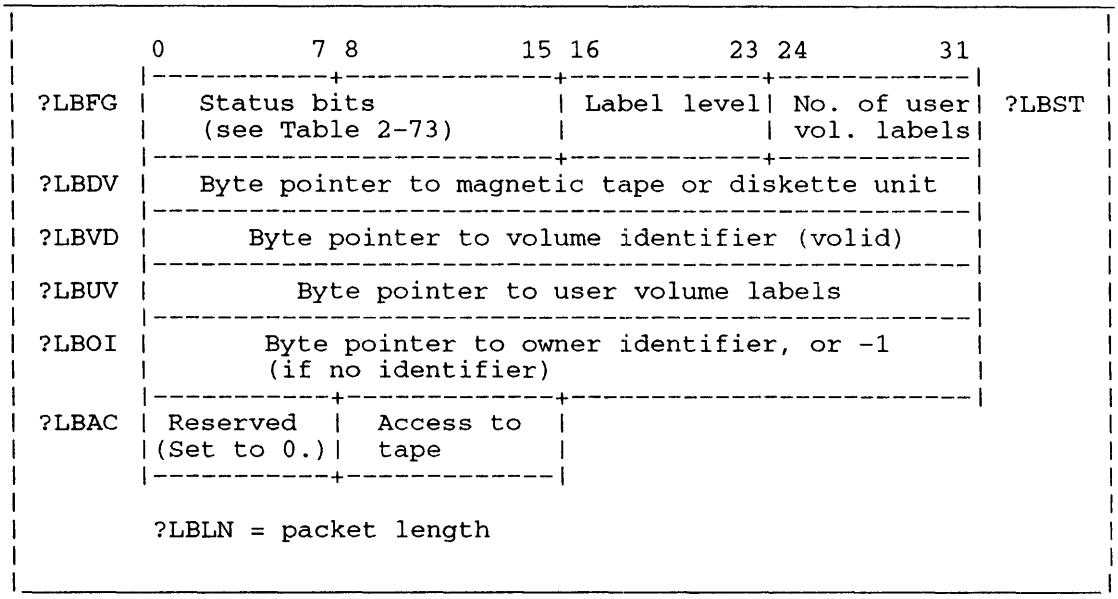


Figure 2–91. Structure of ?LABEL Packet

If you default the density mode, the operating system sets the unit to the density specified in the system–generation procedure. Before you default this parameter, make sure the tape's density matches the density specification set during the system–generation procedure.

Parameter ?LBSC in offset ?LBFG ("scratch this tape") causes the operating system to "scratch" (overwrite) all data on the tape or diskette as it supplies the label.

Use the right byte of offset ?LBAC to specify access to the media. Typically, you supply a space character (ASCII 040) in this byte, which gives all users full access to the media.

Table 2–73. Contents of ?LABEL Packet*

Offset	Contents
?LBFG	Status bits ?LBIM--Label is in IBM format (tape only). ?LBSC--Scratch this tape/diskette. ?LBMF--Use buffered mode (MTJ Model 6352 tape only). ?LBMS--Use streaming mode (MTJ Model 6352 tape only). Density mode (tape only): ?LB8--800 bytes/inch. ?LB16--1600 bytes/inch. ?LB62--6250 bytes/inch. ?LBAM--automatic density matching. ?LB5--Low tape density. ?LB6--Medium tape density. ?LB7--High tape density. Error handling (diskette only): ?LBMR--Check for bad blocks and remap if necessary. ?LBMP--Check for bad blocks and return ERBDK if any are present.
?LBST	Left byte: label level. Supply 1, 2, 3, or 4; the default is 3. Right byte: number of user volume labels (tape only). Supply 1, 2, 3, ..., or 9.
?LBDV (doubleword)	Byte pointer to magnetic tape or diskette unit.
?LBVD (doubleword)	Byte pointer to volume identifier (volid).
?LBUV (doubleword)	Byte pointer to user volume labels (tape only). You can have a maximum of 76 bytes in these labels and you must separate the labels with the null character.
?LBOI (doubleword)	Byte pointer to owner identifier or -1 (if no identifier).
?LBAC	Left byte: reserved. (Set to 0.) Right byte: accessibility (access to tape).

*There is no default unless otherwise specified.

Sample Packet

The following sample packet labels a tape so that everyone can use it.

```
PKT:    .BLK    ?LBLN           ;Allocate enough space for packet.
                                ;Packet length = ?LBLN.
        .LOC    PKT+?LBFG       ;Status bits and/or density mode.
        .WORD   ?LBAM           ;Set correct density automatically.
        .LOC    PKT+?LBST       ;Left byte: label level; right byte:
                                ;number of user volume label.
        .WORD   3*400+1         ;Level = 3, 1 user volume label.
        .LOC    PKT+?LBDV       ;Byte pointer to magnetic tape unit.
        .DWORD  MAG*2           ;Byte pointer to MAG.
        .LOC    PKT+?LBVD       ;Byte pointer to volid.
        .DWORD  VOL*2           ;Byte pointer to VOL.
        .LOC    PKT+?LBUV       ;Byte pointer to user volume labels.
        .DWORD  UVL*2           ;Byte pointer to UVL.
        .LOC    PKT+?LBOI       ;Byte pointer to owner identifier.
        .DWORD  -1              ;No owner identifier.
        .LOC    PKT+?LBAC       ;Access to tape.
        .WORD   <40>            ;Everyone can use it (ASCII 40 =
                                ;space).
        .LOC    PKT+?LBLN       ;End of packet.
```

Notes

- See the description of ?OPEN in this chapter.
- For information on magnetic tape density, refer to the chapter about VSGEN in the manual *Installing, Starting, and Stopping AOS/VS II.*
- Refer to the section about using the LABEL utility in the manual *Managing AOS/VS and AOS/VS II.*

AOS/VS II only

?LDUINFO *[packet address]*
error return
normal return

Input		Output	
AC0	Reserved (set to 0)	AC0	Unchanged or error code
AC1	Reserved (set to 0)	AC1	Unchanged
AC2	The system call packet address, unless you specify the address as an argument to ?LDUINFO	AC2	Unchanged

Error Codes in AC0

EREOF	End of file
ERPKT	Illegal packet identifier
ERRVN	Reserved value not zero
ERVBP	Invalid byte pointer passed as as system call argument
ERPRE	Invalid system call parameter
ERIFT	Illegal file type
ERFAD	File access denied
ERLDX	LDU does not exist
ERFDE	File does not exist
ERRAD	Read access denied

Why Use It

You use ?LDUINFO to obtain information about the logical disk units in your system. It returns the information necessary to help manage your LDU's both mirrored and non–mirrored.

Who Can Use It

?LDUINFO requires read access to the target LDU's root if issued against an initialized logical disk unit. If issued against a physical disk, it requires read and execute access to the physical disk.

What It Does

?LDUINFO returns information pertaining to a physical disk or an initialized logical disk unit. You frequently use ?LDUINFO along with ?XINIT, ?MIRROR, and ?RELEASE since they also operate on logical disks.

The call returns information regarding an entire logical disk or only a given set of images. It can also examine a physical disk and return

- All of the logical disk information available on the physical disk.
- Only that information pertaining to a specific logical disk.
- Information pertaining to a specified logical disk and one or more of it's images.

The ?LDUINFO system call uses a main packet and one of two subpackets. Use the main packet and the ?LDU_PKT subpacket to obtain information about an initialized logical disk. Use the main packet and the ?PIECE_PKT subpacket to obtain information about a physical disk.

The ?LDUINFO main packet contains all of the information used by the different ?LDUINFO functions. Specialization occurs within the subpackets. Figure 2–92 and Table 2–74 describe the contents of the main packet.

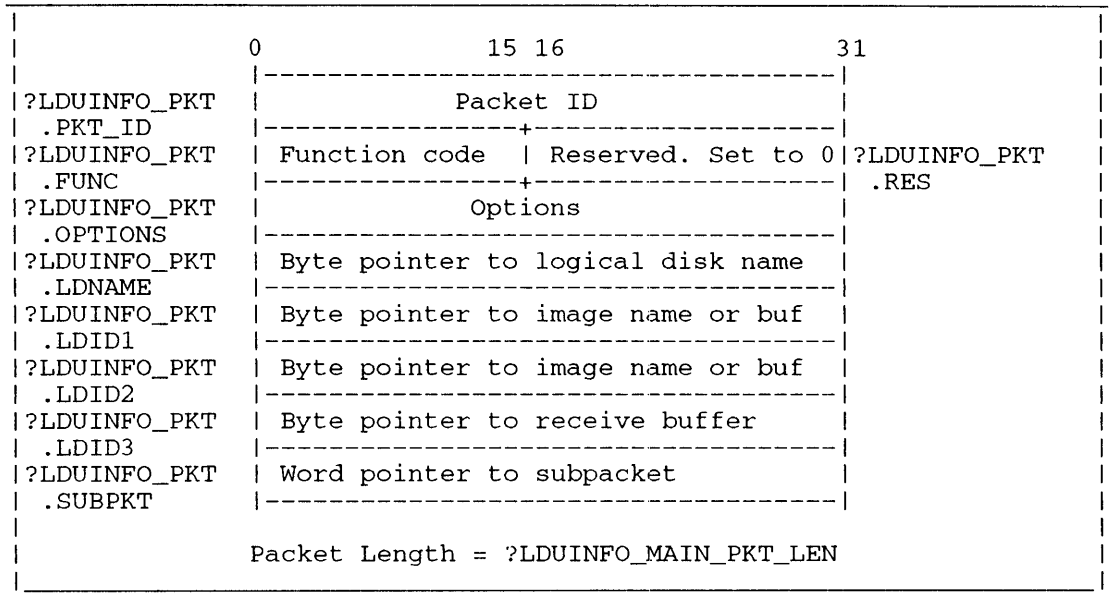


Figure 2–92. Structure of ?LDUINFO Main Packet

Table 2–74. Contents of ?LDUINFO Main Packet

Offset	Contents
?LDUINFO_PKT .PKT_ID	Packet Identifier (Set to ?LDUINFO_PKT_PKTID.)
?LDUINFO_PKT .FUNC	Function Code. Contains one of the following values: ?LDUINFO_GET_LDU_INFO obtains information for an initialized LDU. Requires a byte pointer in offset ?LDUINFO_PKT.LDNAME. ?LDUINFO_GET_PIECE_INFO obtains information for a physical disk. Requires additional information in offset ?LDUINFO_PKT.LDNAME.

(continued)

Table 2–74. Contents of ?LDUINFO Main Packet

Offset	Contents
?LDUINFO_PKT .RES	Reserved. (Set to 0.)
?LDUINFO_PKT .OPTIONS	Options Word. Contains the following flags. ?LDUINFO_LDNAME_SPECIFIED indicates that an LDU must be set when using function code ?LDUINFO_GET_LDU_INFO. ?LDUINFO_LDID_SPECIFIED indicates that one or more logical disk image ID is specified. Requires a byte pointer in offset ?LDUINFO_PKT.LDNAME.
?LDUINFO_PKT .LDNAME	For function code ?LDUINFO_GET_LDU_INFO and option ?LDUINFO_LDID_SPECIFIED, this offset contains a byte pointer to a null-terminated logical disk pathname. For function code ?LDUINFO_GET_PIECE_INFO, initialize this offset to 0 or with a byte pointer to a buffer of size ?MXFN.
?LDUINFO_PKT .LDID1	Contains a byte pointer to a null-terminated logical disk image name if option ?LDUINFO_LDID_SPECIFIED is set. Otherwise the offset contains a byte pointer to a buffer of size ?MXFN.
?LDUINFO_PKT .LDID2	Contains a byte pointer to a null-terminated logical disk image or 0 if option ?LDUINFO_LDID_SPECIFIED is set. Otherwise the offset contains a byte pointer to a buffer of size ?MXFN.
?LDUINFO_PKT .LDID3	Contains 0 if option ?LDUINFO_LDID_SPECIFIED is set. Otherwise it contains a byte pointer to a buffer of size ?MXFN.
?LDUINFO_PKT .SUBPKT	Contains a word pointer to a packet of type ?LDU_PKT if ?LDUINFO_PKT.FUNC contains ?LDUINFO_GET_LDU_INFO. Contains a word pointer to a packet of type ?PIECE_INFO if ?LDUINFO_PKT.FUNC contains ?LDUINFO_GET_PIECE_INFO.

(concluded)

?LDU_PKT Subpacket

The ?LDU_PKT subpacket contains fields for all of the information returned when ?LDUINFO is issued against an initialized logical disk. Figure 2–93 and Table 2–75 describe the contents of the ?LDU_PKT subpacket.

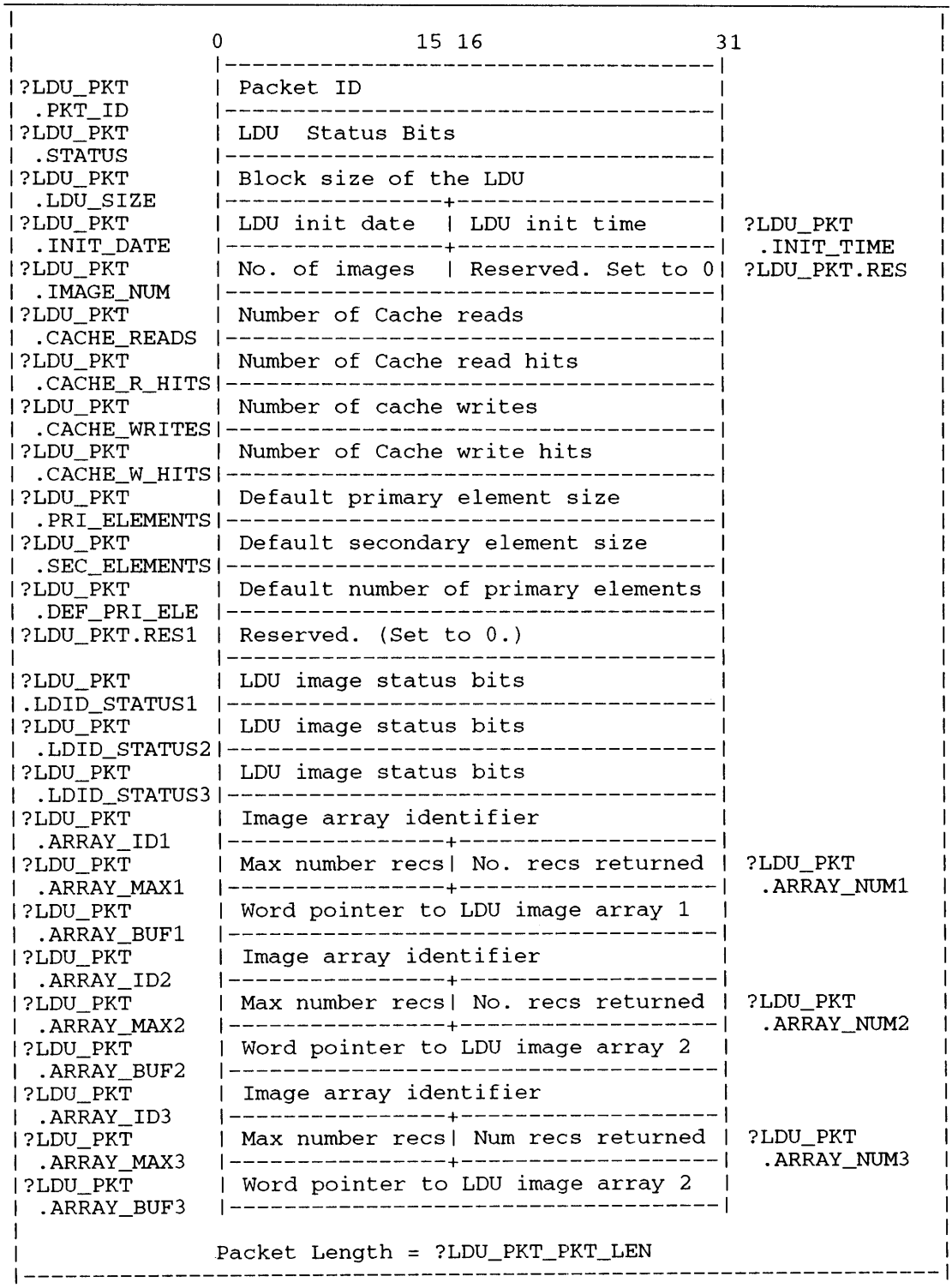


Figure 2–93. Structure of ?LDU_PKT Subpacket

Table 2–75. Contents of ?LDU_PKT Subpacket

Offset	Contents
?LDU_PKT .PKT_ID	Packet Identifier (Set to ?LDU_PKT_PKTID.)
?LDU_PKT .STATUS	Status Word Contains the following bits as returned values: *?LDU_MIRRORED - indicates that the LDU is mirrored *?LDU_MIRROR_BEING_SYNCHRONIZED indicates that the LDU is in the process of being synchronized *?LDU_IMAGE_REMOVED - indicates that one or more images was removed from the LDU
?LDU_PKT .LDU_SIZE	Returns the block size of the LDU.
?LDU_PKT .INIT_DATE	Date the LDU was last initialized.
?LDU_PKT .INIT_TIME	Time the LDU was last initialized.
?LDU_PKT .IMAGE_NUM	Returns the number of images in the LDU.
?LDU_PKT .CACHE_READS	The number of cache reads on this LDU.
?LDU_PKT .CACHE_R_HITS	The number of times that a read hit occurred on this LDU.
?LDU_PKT .CACHE_WRITES	The number of cache writes on this LDU.
?LDU_PKT .CACHE_W_HITS	The number of times that a write hit occurred on this LDU.
?LDU_PKT .PRI_ELEMENTS	Default primary element size.
?LDU_PKT .SEC_ELEMENTS	Default secondary element size.
?LDU_PKT .DEF_PRI_ELE	Default number of primary elements.
?LDU_PKT .RES1	Reserved. (Set to 0.)

(continued)

Table 2–75. Contents of ?LDU_PKT Subpacket

Offset	Contents
?LDU_PKT .LDID_STATUS1	Returns logical disk image status information for the first image. The defined status bits are: *?LDU_IMAGE_HARDWARE_MIRRORED - indicates image is hardware mirrored *?LDU_PRIMARY_IMAGE - indicates that this is the primary image
?LDU_PKT .LDID_STATUS2	Returns logical disk image status information for the second image. The defined status bits are the same as above.
?LDU_PKT .LDID_STATUS2	Returns logical disk image status information for the second image. The defined status bits are the same as above.
?LDU_PKT .ARRAY_ID1	Place ?LDU_PKT_ARRAYID here.
?LDU_PKT .ARRAY_MAX1	Place the size of the array given here. An array of size ?LDU_PKT_ARRAY_MAX is recommended. This array will contain records each of which will contain a byte pointer to a buffer of size ?MXFN to hold the disk name along with three doubleword fields for the size of the piece, the starting LDA of the piece, and the size of the bad block table for that piece. Note that ?LDU_PKT_ARRAY_MAX refers to record sizes and not word sizes. Each record will be ?LDU_PKT_ARRAY_REC_LEN words long.
?LDU_PKT .ARRAY_NUM1	Returns the number of actual array elements initialized with information.
?LDU_PKT .ARRAY_BUF1	Contains a 32 bit pointer to the buffer which will hold the array. The buffer must be ?LDU_PKT.ARRAY_MAX1 * ?LDU_PKT_ARRAY_WORD_SIZE words long.
?LDU_PKT .ARRAY_ID2	Place ?LDU_PKT_ARRAYID here.
?LDU_PKT .ARRAY_MAX2	Place the size of the array given here. This array is for the secondary image. If ?LDUINFO_LDID_SPECIFIED is set and the second LDID is not specified, this and the following three fields may be set to 0.

(continued)

Table 2–75. Contents of ?LDU_PKT Subpacket

Offset	Contents
?LDU_PKT .ARRAY_NUM2	Returns the number of actual array elements initialized with information.
?LDU_PKT .ARRAY_BUF2	Contains a 32 bit pointer to the buffer which will hold the array, or 0.
?LDU_PKT .ARRAY_ID3	Place ?LDU_PKT_ARRAYID here.
?LDU_PKT .ARRAY_MAX3	Place the size of the array given here. This array is for the tertiary image. If ?LDUINFO_LDID_SPECIFIED is set and the third LDID is not specified, this and the following three fields may be set to 0.
?LDU_PKT .ARRAY_NUM3	Returns the number of actual array elements initialized with information.
?LDU_PKT .ARRAY_BUF3	Contains a 32 pointer to the buffer which will hold the array, or 0.

(concluded)

?LDU_PKT Array Record Definition

Figure 2–94 shows the array record for arrays of type ?LDU_PKT_ARRAYID.

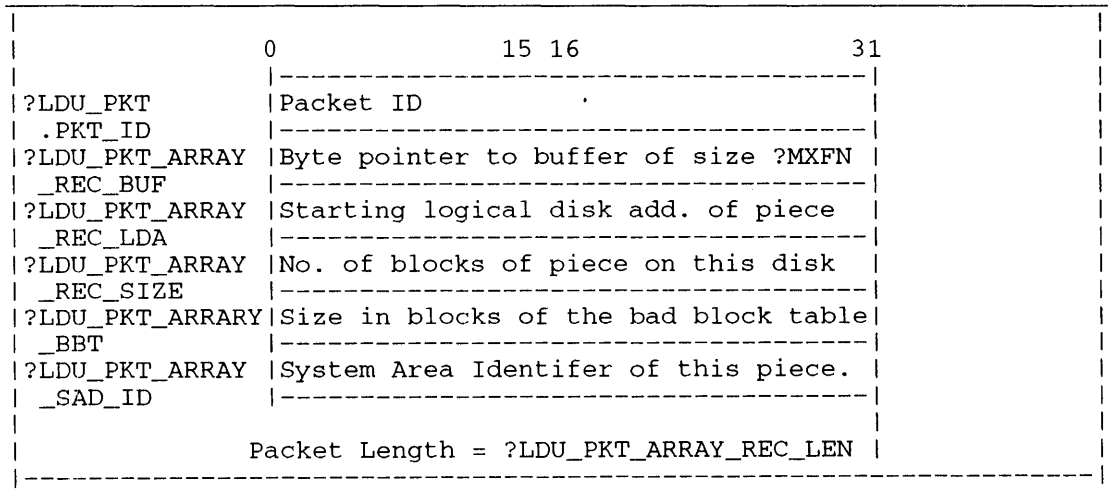


Figure 2–94. Structure of ?LDU_PKT Array Record

?PIECE_PKT Subpacket

The ?PIECE_PKT Subpacket contains fields for all of the information returned when ?LDUINFO is issued against a physical disk. Figure 2–95 and Table 2–76 describe the contents of the ?PIECE_PKT subpacket.

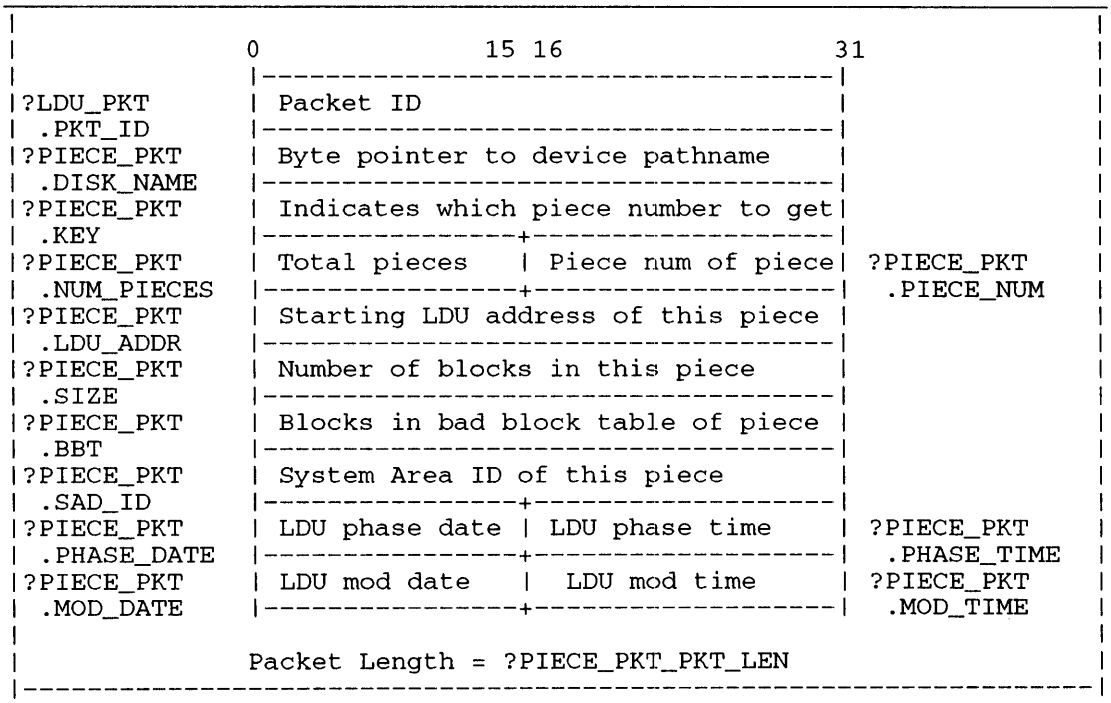


Figure 2–95. Structure of ?PIECE_PKT Subpacket

Examples

This section contains some examples of how to use the ?LDUINFO call.

Find the Mirror Status of an LDU

Issuing ?LDUINFO with the ?LDUINFO_GET_LDU_INFO function code returns mirroring status. ?LDUINFO_LDNAME_SPECIFIED must be set and ?LDUINFO_PKT.LDNAME must be initialized. ?LDU_PKT.LDID<1,2,3> should all contain byte pointers to buffers of size ?MXFN and ?LDUINFO_PKT.SUBPKT should contain a word pointer to a packet of type ?LDU_PKT.

Inside of the ?LDU_PKT Subpacket, ?LDU_PKT.ARRAY_ID<1,2,3> should be set. ?LDU_PKT.ARRAY_BUF<1,2,3> should all contain word pointers to buffers big enough to hold ?LDU_PKT_ARRAY_REC_SIZE * the value placed in ?LDU_PKT.ARRAY_MAX<1,2,3>. Additionally, each array record should have it's first field initialized with a byte pointer to a buffer of size ?MXFN.

Table 2–76. ?PIECE_PKT Subpacket Contents

Offset	Contents
?PIECE_PKT .PKT_ID	Packet Identifier (Set to ?PIECE_PKT.PKTID.)
?PIECE_PKT .DISK_NAME	Byte pointer to the disk device pathname.
?PIECE_PKT .KEY	Key value. Used by the operating system and should be set to 0 the first time the call is issued against a given physical disk.
?PIECE_PKT .NUM_PIECES	Returns the number of pieces associated with the LDU image.
?PIECE_PKT .PIECE_NUM	Returns the number of this piece within the LDU image.
?PIECE_PKT .LDU_ADDR	Returns Starting address of this logical disk unit.
?PIECE_PKT .SIZE	Returns total number of blocks in this piece.
?PIECE_PKT .BBT	Returns size of the bad block table (in blocks) for this piece.
?PIECE_PKT .SAD_ID	System area identifier for this piece. Useful when you use the GOPEN system call on the piece.
?PIECE_PKT .PHASE_DATE	Returns phase date used to keep track of most up-to-date mirror image.
?PIECE_PKT .PHASE_TIME	Returns phase time used to keep track of most up-to-date mirror image.
?PIECE_PKT .MOD_DATE	Returns date of first modification after init or mirror break.
?PIECE_PKT .MOD_TIME	Returns time of first modificaton after init or mirror break

When the system call returns, the system fills the buffers specified in the ?LDUINFO_LDID<1,2,3> with the first, second, and third image names in the LDU. The second or third images may not exist, in which case the buffers are unchanged. If the LDU is mirrored (second image exists — third image may exist) the call returns ?LDU_MIRRORED set in ?LDU_PKT.STATUS. If synchronizing one or more of the images, then the call also sets ?LDU_MIRROR_BEING_SYNCHRONIZED.

For each piece of each image, the call initializes an array record. For the first image, the information is in the buffer specified in ?LDU_PKT.ARRAY_BUF1 and so on. There is one array record for each piece. ?LDU_PKT.ARRAY_NUM<1,2,3> contains the number of initialized records (number of pieces). Each record contains a byte pointer to a buffer that has been initialized with the disk name (@DPXX, etc). The record also contains the size of the piece and the first logical disk address that can be found on that piece.

Additionally, the ?LDU_PKT.LDID_STATUS<1,2,3> fields indicate for each defined image whether or not it is hardware mirrored and whether or not it is the primary image. The hardware–mirror–returned bit is set for both hardware mirror images.

Determining If a Given Image Is Hardware Mirrored

Issue ?LDUINFO as above except set the ?LDUINFO_LDID_SPECIFIED bit. The ?LDUINFO_PKT.LDID1 field should contain a byte pointer to a null–terminated image name.

In this situation, the ?LDUINFO_PKT.LDID<2,3> fields are not changed and neither are the array records for the arrays identified in ?LDU_PKT.ARRAY_BUF<2,3> nor the status bits in ?LDU_PKT.LDID_STATUS<2,3>. In fact, all of the above fields are validated to zero.

If you are truly interested in only the hardware mirror status, you need only check the ?LDU_PKT_LDID_STATUS1 field. The array records for the specified image are filled, but of course, you may ignore this data.

If you are interested in two images, use the ?LDUINFO_LDID2 field to specify the additional LDU image. Initialize the other corresponding output fields.

Identifying All of the LDU Pieces on a Physical Disk

Issuing ?LDUINFO with the ?LDU_GET_PIECE_INFO system call identifies all of the LDU pieces on a physical disk. For the first call, set the ?PIECE_PKT.KEY field to zero. For the first and all subsequent calls, set the ?PIECE_PKT.DISK_NAME field to point to a null–terminated device name. As this call only returns the information for one piece at a time, issue the call continuously until the call returns an EREOF (end of file) error. This interface matches that of ?GNFN for directories.

Set ?LDUINFO_PKT.LDNAME and ?LDUINFO_PKT.LDID1 to point to buffers of size ?MXFN.

When the call returns, the buffers pointed to by ?LDUINFO_PKT.LDID1 and ?LDUINFO_PKT.LDNAME contain the logical disk image name and logical disk name respectively. The remainder of the ?PIECE_PKT subpacket is filled in as well.

Determining If LDU 'X' Has a Piece on a Given Disk

Issue ?LDUINFO as above, except set the ?LDUINFO_PKT.LDNAME field to point to a null–terminated LDU name. Also set the ?LDUINFO_LDNAME_SPECIFIED bit.

The call returns EREOF, or the appropriate information into the ?PIECE_PKT.

To determine if a given LDU image has a piece on a certain disk, perform the above actions and initialize ?LDUINFO_PKT.LDID and set ?LDUINFO_LDID_SPECIFIED.

?LEFD
error return
normal return

Input

None

Output

AC0 Contents lost

AC1 Undefined

AC2 Undefined

Error Codes in AC0

No error codes are currently defined.

Why Use It?

By default, each process begins executing with the CPU in LEF mode. For a process to perform I/O, LEF mode must be disabled (and I/O mode enabled). Thus, you must issue ?LEFD before you issue primitive I/O instructions; that is, I/O instructions distinct from the I/O system calls.

Who Can Use It?

There are no special process privileges needed to issue this call, and there are no restrictions concerning file access.

What It Does

?LEFD disables LEF mode (load–effective address mode) for the calling process. When LEF mode is disabled, the load–effective address instructions execute as I/O instructions. If LEF mode is already disabled when you issue ?LEFD, ?LEFD takes the normal return, but has no effect.

Note that although ?LEFD disables LEF mode, it does not automatically re–enable I/O mode. To disable LEF mode and re–enable I/O mode, issue ?DEBL.

Notes

- See the descriptions of ?LEFE, ?LEFS, and ?DEBL in this chapter.

?LEFE
error return
normal return

Input

None

Output

AC0	Contents lost
AC1	Undefined
AC2	Undefined

Error Codes in AC0

No error codes are currently defined.

Who Can Use It?

There are no special process privileges needed to issue this call, and there are no restrictions concerning file access.

What It Does

?LEFE enables LEF mode (load–effective address mode) for the calling process. LEF mode allows the caller to successfully issue LEF instructions. If LEF mode is already enabled when you issue ?LEFE, ?LEFE takes the normal return, but has no effect.

Why Use It?

?LEFE is the inverse of ?LEFD. You must issue ?LEFE if you want to re–enable LEF mode after you disable it with ?LEFD.

Notes

- See the descriptions of ?LEFD, ?LEFS, and ?DDIS in this chapter.

?LEFS — Returns the current LEF mode status.

?LEFS
error return
normal return

Input

None

Output

AC0 One of the following:

- –1 if LEF mode is enabled
- 0 if LEF mode is disabled

AC1 Undefined

AC2 Undefined

Error Codes in AC0

No error codes are currently defined.

Who Can Use It?

There are no special process privileges needed to issue this call, and there are no restrictions concerning file access.

What It Does

?LEFS returns the current LEF–mode (load–effective address mode) status to AC0. If the CPU is in LEF mode, AC0 contains –1 on output. If the CPU is not in LEF mode, AC0 contains 0 on output.

No device I/O can occur while the CPU is in LEF mode, because LEF–mode instructions and I/O instructions use the same bit patterns. Similarly, when LEF mode is disabled, the operating system executes LEF instructions as if they are I/O instructions. Therefore, the state of the CPU determines whether the operating system executes LEF–mode or I/O–mode instructions.

Why Use It?

You can use ?LEFS to check the status of LEF mode.

Notes

- See the descriptions of ?LEFD and ?LEFE in this chapter.

?LMAP
error return
normal return

Input

AC0	Source address
AC1	Destination address
AC2	Reserved (Set to 0.)

Output

AC0	Unchanged
AC1	Unchanged
AC2	Unchanged

Error Codes in AC0

ERSNR	Source not resident
ERDNR	Destination not resident
ERPRV	Caller not privileged
ERMPR	Not an unshared page
ERPTY	Illegal process type

Why Use It?

?LMAP allows privileged programs to monitor system counters and other system status information.

Who Can Use It?

To use ?LMAP, Superprocess mode must be turned on. The issuer of ?LMAP must be resident at the time the call is issued and must remain resident while the ?LMAP is in effect. There are no restrictions concerning file access.

What It Does

?LMAP allows a privileged user to map the page containing the process's source address into the page containing the destination address.

The source page can be in any ring of your address space, but it must be in memory (resident) at the time the ?LMAP call is issued, and, if the source address is not in Ring 0, the page must be wired. The source page cannot change its status (residency, wired/nonwired, current usage, etc.) in any way while it remains the source for the ?LMAP. It is up to the ?LMAP caller to ensure that its status does not change, as no record is made that the ?LMAP was performed.

The destination page for the ?LMAP must be an unshared, wired page when ?LMAP is issued. The system call removes the destination page from the working set of the process and replaces it with a read–only mapping of the source page. The ?LMAP destination page must not be referred to by any subsequent system call. It is up to the ?LMAP caller to ensure this, as there is no information in the system that the destination address is an ?LMAP page.

CAUTION: Failure to heed these restrictions may cause the operating system to PANIC or HANG. These potential problems restrict ?LMAP to Superprocess privilege holders.

AOS/VS

?LOCALITY *[packet address]*
error return
normal return

Input

AC0	Reserved (Set to 0.)
AC1	Reserved (Set to 0.)
AC2	Address of the ?LOCALITY packet, unless you specify the address as an argument to ?LOCALITY

Output

AC0	Unchanged
AC1	Unchanged
AC2	Address of the ?LOCALITY packet

Error Codes in AC0

ERILV	Illegal locality value
ERPRH	Process not in hierarchy
ERPRV	Caller not privileged for this action

Why Use It?

Use this system call to change the user locality and class of a process.

Who Can Use It?

You must have the following privileges in order to issue ?LOCALITY:

- If you have the system manager privilege ON, you may change the locality of any process on the system to any legal user locality value.
- If you do not have the system manager privilege ON, then
  - You may only issue ?LOCALITY against yourself or processes in your subtree (i.e., sons, grandsons, etc.).
  - Also, you may only change processes to those user localities designated at ?PROC time as legal for your process.

There are no restrictions concerning file access.

What It Does

This system call allows a process to change the user locality (and thus class) of itself or of another process. When you change a process's user locality, AOS/VS indexes into the class matrix and calculates the new class ID for the process. AOS/VS then schedules the process on the basis of its new class.

You modify user locality for the calling process as follows. Place the value of ?LOC_PID in offset ?LOC_PKT.PCODE of the packet, and place –1 in offset ?LOC_PKT.PID.

You modify user locality for a different process in one of the two following ways:

- Assume that you know the name of the process. Place the value of ?LOC_PNAME in offset ?LOC_PKT.PCODE. Place in offset ?LOC_PKT.PNAME a byte pointer to the process name. Set offset LOC_PKT.PID to zero.
- Assume that you know the PID of the process. Place the value of ?LOC_PID in offset ?LOC_PKT.PCODE. Place zero in offset ?LOC_PKT.PNAME. Place in offset ?LOC_PKT.PID the PID.

Figure 2–96 shows the structure of the ?LOCALITY parameter packet, and Table 2–77 describes its contents.

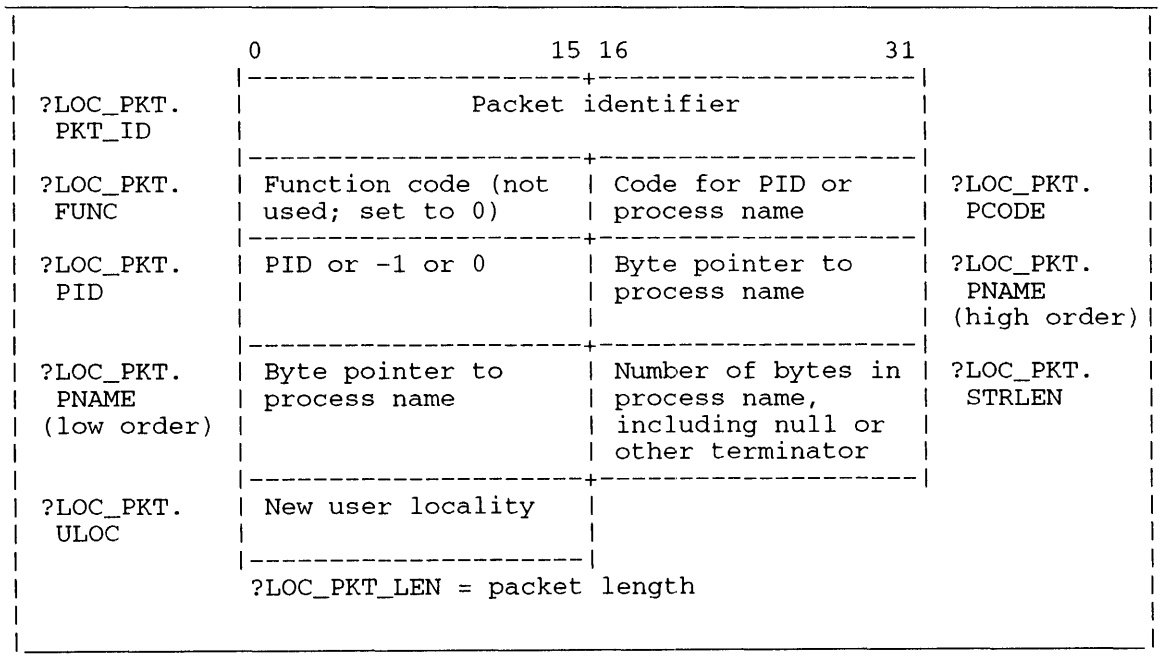


Figure 2–96. Structure of ?LOCALITY Packet

Table 2–77. Contents of ?LOCALITY Packet

Offset	Contents
?LOC_PKT.PKT_ID (doubleword)	Packet identifier. Place ?LOC_PKT_PKTID here.
?LOC_PKT.FUNC	Function code. Not used. (Set to 0.)
?LOC_PKT.PCODE	Code word into which you place ?LOC_PID when you supply a PID or else ?LOC_PNAME when you supply a process name.
?LOC_PKT.PID	Supply -1 for the calling process. For another process, place its PID here or else 0 here and the process's name information in the next two offsets.
?LOC_PKT.PNAME (doubleword)	If you've placed 0 in the previous offset, then place a byte pointer here to the process's name and place its length in the next offset.
?LOC_PKT.STRLEN	If you've placed 0 in offset ?LOC_PKT.PID and a byte pointer in the previous offset, then place the number of bytes in the byte pointer here.
?LOC_PKT.ULOC	Supply the new user locality.

?LOGCALLS — Logs system calls.

?LOGCALLS
error return
normal return

Input

AC0	Contains –1
AC1	Flag word: • ?LSTART — start system call logging • ?LFOP — start forced output
AC2	Byte pointer to pathname of file to create (ignored if ?LSTART in AC1 = 0)

Output

AC0	Unchanged
AC1	Unchanged
AC2	Unchanged

Error Codes in AC0

ERVBP Invalid byte pointer passed as a system call argument
File system error codes

Why Use It?

?LOGCALLS allows you to keep track of every system call that your process issues, including invalid system calls and system calls that inner rings issue.

You can use system call logging both as a debugging tool and to help you analyze a program's performance.

Who Can Use It?

There are no special process privileges needed to issue this call, and there are no restrictions concerning file access.

What It Does

?LOGCALLS logs system calls (including invalid system calls) into a file whose name you specify in AC2. To start logging, issue ?LOGCALLS with the ?LSTART flag set in AC1.

When you issue ?LOGCALLS to start logging after logging has already begun, the operating system closes the old log file and creates a new one. However, if you specify a file that already exists as a log file, ?LOGCALLS tries to delete the old file and create a new one with the same name. This allows you to periodically restart system call logging so that your log file contains only those system calls that you issued recently.

You can stop system call logging voluntarily by issuing ?LOGCALLS with the ?LSTART bit set to off in AC1. When a program terminates (via ?TERM to your process, ?RETURN, ?BRKFL, Ctrl–C Ctrl–B, or if a superior process terminates your process), system call logging stops involuntarily. No matter how system call logging stops, you can still read the log file.

If you try to stop system call logging, but it has already stopped or was never started, the operating system ignores all but the first ?LOGCALLS stop and returns normally.

You can minimize system write activity by setting the ?LFOP bit to 0 in AC1. This causes the operating system to buffer all output before it writes it to the log file. However, if the process is terminated without the ?LFOP bit set in AC1, the operating system may not write the last buffer of system calls to the log file. Therefore, if you want to make sure that the log file contains each system call when it is made, you can force output to the file without buffering by setting the ?LFOP bit in AC1.

To read the log file, you can use the LOGCALLS utility program, which prints the output in report form, or you can write your own program.

The system call log file consists of the following elements:

- Header record.

  The operating system creates the header record when system call logging starts.

- Detailed records for each system call.

  The record format is dynamic (?RTDY). Therefore, you must specify the length of the record that you want to read. To specify binary read, set the ?IBIN flag in the ?ISTI offset of the ?OPEN/?READ packet.

Header record entries in the system call log file contain ?LOGHREC words in the following format (offsets are in octal):

Words 0 through 5	Time of day that logging began
Words 0 through 1	Seconds
Words 2 through 3	Minutes
Words 4 through 5	Hours
Words 6 through 13	Day that logging began
Words 6 through 7	Day of month
Words 10 through 11	Month of year
Words 12 through 13	Year
Words 14 through 23	Process name in the following format: username:simple_processname The process name contains up to 15 characters and a null (ASCII 0) terminating character.
Word 24	Flag word Bit ?LOGF16U set means 16–bit process. Other bits are undefined.

Words 25 through 224	Pathname of .PR file that was loaded when you started system call logging. This pathname has up to 255 characters and a null (ASCII 0) terminating character.

The detailed record entries contain ?LOGDREC words in the following format (offsets are in octal):

Words 0 through 1	Contents of caller's AC0
Words 2 through 3	Contents of caller's AC1
Words 4 through 5	Contents of caller's AC2
Words 6 through 7	Location of the system call word (PC)
Words 10 through 11	Contents of caller's frame pointer
Word 12	Task identifier (TID)
Word 13	Unique TID
Word 14	System call number

Notes

- See the descriptions of ?RETURN, ?BRKFL, ?OPEN, and ?READ in this chapter.

?LOGEV — Enters an event in the system log file.

AOS/VS

?LOGEV
error return
normal return

Input

AC0	Event code you want to enter in the log file
AC1	Byte length of the message you want to enter in the log file
AC2	Byte pointer to the message you want to enter in the log file (ignored, if AC1 contains 0)

Output

AC0	Unchanged
AC1	Unchanged
AC2	Unchanged

Error Codes in AC0

ERPRP	Illegal priority
ERVBP	Invalid byte pointer passed as a system call argument

Why Use It?

You can use ?LOGEV with ?SYLOG, which creates, renames, or checks the status of the system log file.

Who Can Use It?

Access to both ?LOGEV and ?SYLOG is restricted. Only a process with the Superuser privilege can issue ?LOGEV; only the operator process (PID 2) can issue ?SYLOG. There are no restrictions concerning file access.

What It Does

?LOGEV allows a process with the Superuser privilege to enter an event code and/or a message into the system's log file, :SYSLOG. The system log file contains current information about all user terminal or batch processes, such as usernames, the times each user logged on and off, the devices each user accessed, and information about CPU and memory usage.

The operating system enters the log file data in the format shown in Figure 2–97.

The operating system records the event code, the length of the entire log entry, and the date and time it was entered in a packet of eight 16–bit words. However, the message text, if you choose to use it, can be any length up to 496 characters. If necessary, the operating system pads the message text with zeros to a multiple of eight 16–bit words.

You can use the event–code field to assign each event a unique code, from ?LUMI through ?LMAX. Event codes ?LSMI and ?LDMA are reserved for system use.

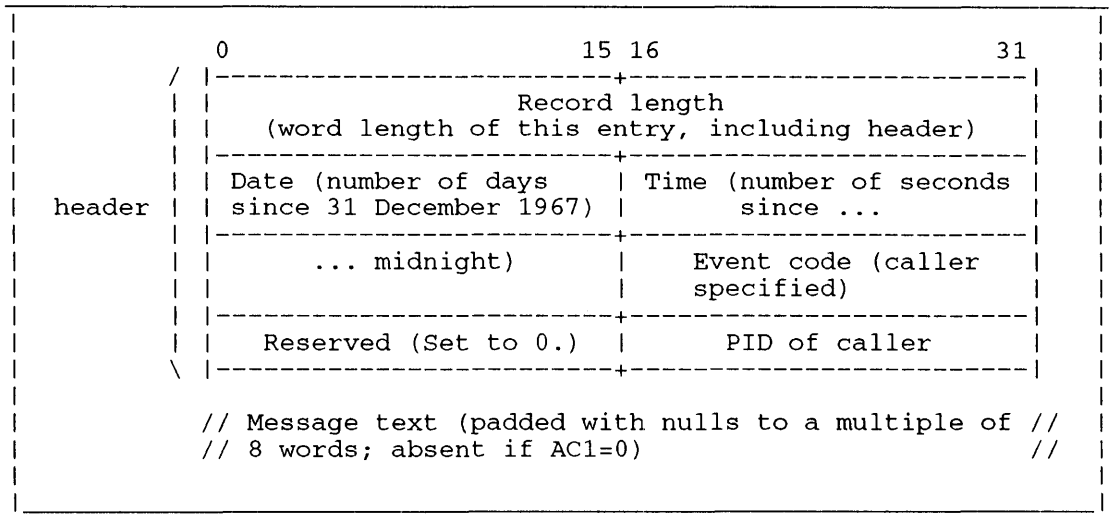


Figure 2–97. ?LOGEV Event Logging Format

The following restrictions apply to ?LOGEV:

- If AC1 (message length) = 0, the operating system ignores whatever message you specify and writes only the packet to the log file.
- The operating system writes only the first 760 (octal) bytes of the message to the log file. (If the message exceeds 760 (octal) bytes, the operating system ignores the remainder.)
- If the log function is turned off, the operating system ignores the message, but takes the ?LOGEV normal return.

Notes

- See the description of ?SYLOG in this chapter.
- See PARU.32.SR for a complete list of the event codes.
- Refer to the manual *Managing AOS/VS and AOS/VS II* for information on enabling system logging.
- Refer to the manual *AOS/VS System Concepts* for information on the correspondence between each event code and the format of its message text section in the record that ?LOGEV enters in :SYSLOG.

?LPCLASS Gets/sets logical processor class assignments.

AOS/VS

?LPCLASS *[packet address]*
error return
normal return

Input

AC0	Reserved (Set to 0.)
AC1	Reserved (Set to 0.)
AC2	Address of the ?LPCLASS packet, unless you specify the address as an argument to ?LPCLASS

Output

AC0	Unchanged
AC1	Unchanged
AC2	Address of the ?LPCLASS packet

Error Codes in AC0

ERCPC	Invalid class percentage value
ERHLP	Illegal hierarchy level/percentage pair
ERHLT	Nonsequential hierarchy levels designated
ERICD	Illegal function code
ERICI	Invalid class ID
ERITI	Invalid time interval
ERLNE	Logical processor (LP) does not exist
ERPRV	Caller not privileged for this action

Why Use It?

Use this system call to receive the current class scheduling data for a particular LP or to declare class scheduling information for a particular LP.

Who Can Use It?

You must have System Manager privilege to issue this call to set class values, but there are no restrictions concerning file access.

What It Does

To indicate whether you want to get current information about classes for an LP or want to set this information, specify a get/set code in the parameter packet.

The time interval word in the parameter packet specifies the amount of time over which the percentages (in the next words) will be spread. That is, AOS/VS attempts to schedule the various percentages over the specified time interval. The value is between ?LPCL_PKT.TIME_MIN and ?LPCL_PKT.TIME_MAX tenths of a second. At the time of this writing, the respective values were 1 and 100 (0.1 and 10.0 seconds). If you place ?LPCL_PKT.TIME_DEF in this word, you specify the default value of 40*0.1 seconds = 4.0 seconds.

The last portion of the packet is a set of 16 words that forms a table for scheduling each of the 16 possible classes. Each word contains a hierarchy level in its left byte and a percentage value in its right byte. Supplying a percentage value makes the corresponding class primary; AOS/VS attempts to give that class the specified percentage (between 1 and 100) of CPU time. Supplying a hierarchy level makes the corresponding class secondary; the supplied number indicates the hierarchy level of the class. AOS/VS only schedules secondary classes when there are no eligible primary classes ready to run. When this occurs, AOS/VS schedules the secondary classes in order of their hierarchy level (classes at hierarchy level 1 first, classes at hierarchy level 16 last). You cannot supply both a percentage value and a hierarchy level for the same class. That is, a class cannot be both primary and secondary within a single LP.

Be sure that your hierarchy levels are sequential starting with level 1 and increasing in a step of 1. For example, you may designate classes at hierarchy levels 1, 2, 3, and 4 — but not at hierarchy levels 1, 2, 3, and 5.

Figure 2–98 shows the structure of the ?LPCLASS parameter packet, and Table 2–78 describes its contents.

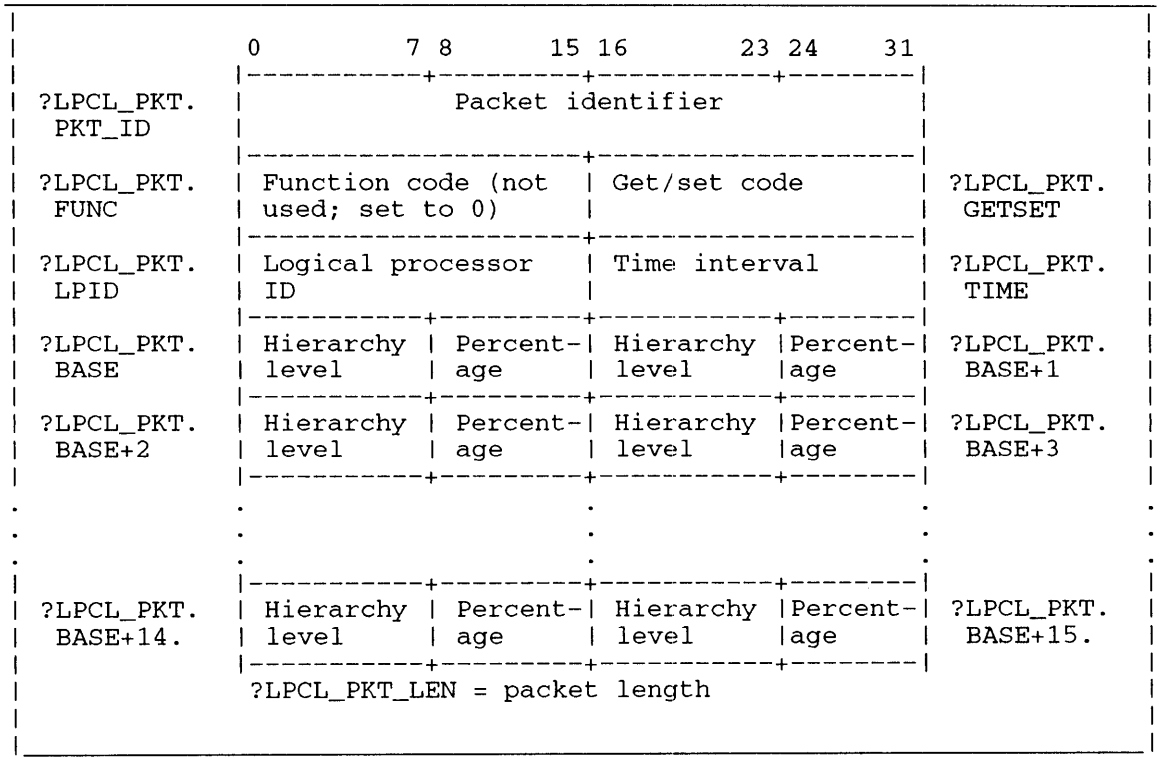


Figure 2–98. Structure of ?LPCLASS Packet

Table 2–78. Contents of ?LPCLASS Packet

Offset	Contents
?LPCL_PKT.PKT_ID (doubleword)	Packet identifier. Place ?LPCL_PKT_PKTID here.
?LPCL_PKT.FUNC	Function code. Not used. (Set to 0.)
?LPCL_PKT.GETSET	Get/set code. You supply ?LPCL_GET to get class values or ?LPCL_SET to set class values.
?LPCL_PKT.LPID	Logical processor ID for the LP about which you will get class data or set class data.
?LPCL_PKT.TIME	Time interval, with the number of tenths of a second, during which you will get or set the schedule of classes.
?LPCL_PKT.BASE	Hierarchy level and percentage of CPU time (left byte/right byte) for class number 0.
?LPCL_PKT.BASE+1	Hierarchy level and percentage of CPU time (left byte/right byte) for class number 1.
?LPCL_PKT.BASE+2	Hierarchy level and percentage of CPU time (left byte/right byte) for class number 2.
?LPCL_PKT.BASE+3	Hierarchy level and percentage of CPU time (left byte/right byte) for class number 3.
.	.
.	.
.	.
?LPCL_PKT.BASE+14.	Hierarchy level and percentage of CPU time (left byte/right byte) for class number 14.
?LPCL_PKT.BASE+15.	Hierarchy level and percentage of CPU time (left byte/right byte) for class number 15.

?LPCREA — Creates a logical processor.

AOS/VS

?LPCREA *[packet address]*
error return
normal return

Input

AC0	Reserved (Set to 0.)
AC1	Reserved (Set to 0.)
AC2	Address of the ?LPCREA packet, unless you specify the address as an argument to ?LPCREA

Output

AC0	Unchanged
AC1	Unchanged
AC2	Address of the ?LPCREA packet

Error Codes in AC0

ERLAI	LP already exists
ERMLP	Attempt to exceed maximum logical processor (LP) count
ERPRV	Caller not privileged for this action

Why Use It?

Use this system call to create a new logical processor (LP).

Who Can Use It?

You must have exclusive System Manager privilege to issue this call, but there are no restrictions concerning file access.

What It Does

This system call creates a new LP. AOS/VS creates an LP and returns its ID number (LPID). Then, you can set up LP class designations, assign job processors, etc.

AOS/VS assigns a value to LPID between ?LPID_MIN and ?LPID_MAX, respectively.

Figure 2–99 shows the structure of the ?LPCREA parameter packet, and Table 2–79 describes its contents.

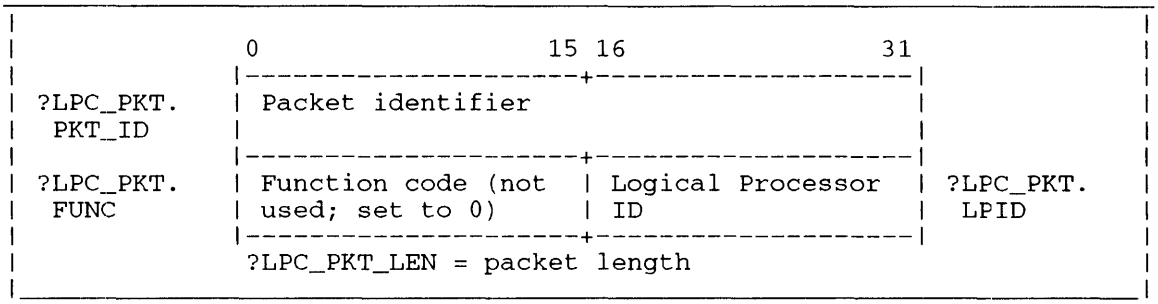


Figure 2–99. Structure of ?LPCREA Packet

Table 2–79. Contents of ?LPCREA Packet

Offset	Contents
?LPC_PKT.PKT_ID (doubleword)	Packet identifier. Place ?LPC_PKT_PKTID here.
?LPC_PKT.FUNC	Function code. Not used. (Set to 0.)
?LPC_PKT.LPID	Logical processor ID. AOS/VS returns this value.

?LPDELE — Deletes a logical processor.

AOS/VS

?LPDELE *[packet address]*
error return
normal return

Input

AC0	Reserved (Set to 0.)
AC1	Reserved (Set to 0.)
AC2	Address of the ?LPDELE packet, unless you specify the address as an argument to ?LPDELE

Output

AC0	Unchanged
AC1	Unchanged
AC2	Address of the ?LPDELE packet

Error Codes in AC0

ERILP	Invalid LPID
ERJAA	Job processors (JP) already attached to logical processor (LP)
ERLNE	LP does not exist
ERLP0	Cannot delete LP 0
ERPRV	Caller not privileged for this action

Why Use It?

Use this system call to remove a logical processor (LP).

Who Can Use It?

You must have exclusive System Manager privilege to issue this call, but there are no restrictions concerning file access.

What It Does

This system call deletes an LP. However, you cannot delete an LP if there are any job processors (JPs) attached to it. You also cannot delete LP 0 — the LP that exists when your AOS/VS system first comes up.

Figure 2–100 shows the structure of the ?LPDELE parameter packet, and Table 2–80 describes its contents.

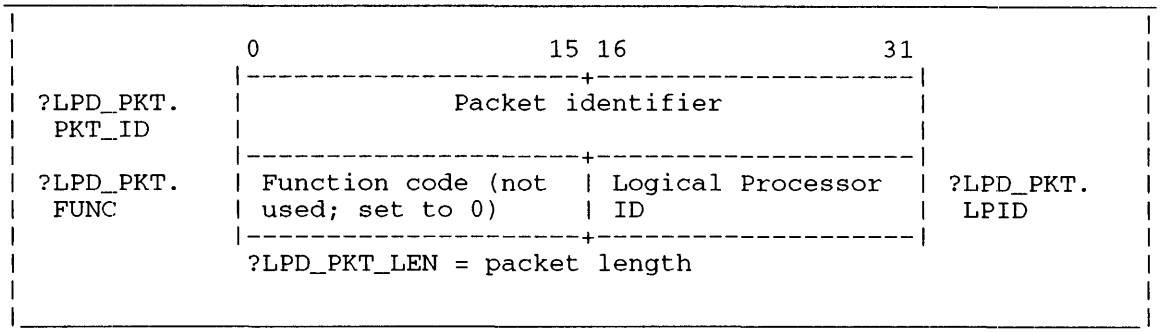


Figure 2–100. Structure of ?LPDELE Packet

Table 2–80. Contents of ?LPDELE Packet

Offset	Contents
?LPD_PKT.PKT_ID (doubleword)	Packet identifier. Place ?LPD_PKT_PKTID here.
?LPD_PKT.FUNC	Function code. Not used. (Set to 0.)
?LPD_PKT.LPID	Logical processor ID. You supply this value.

?LPSTAT — Gets the status of a logical processor.

AOS/VS

?LPSTAT *[packet address]*
error return
normal return

Input

AC0	Reserved (Set to 0.)
AC1	Reserved (Set to 0.)
AC2	Address of the ?LPSTAT packet, unless you specify the address as an argument to ?LPSTAT

Output

AC0	Unchanged
AC1	Unchanged
AC2	Address of the ?LPSTAT packet

Error Codes in AC0

ERICD	Invalid function code
ERILP	Invalid LPID
ERLNE	Logical processor (LP) does not exist

Why Use It?

Use this system call to obtain information about all current logical processors (LP) or about a specific current logical processor.

Who Can Use It?

There are no special process privileges needed to issue this call, and there are no restrictions concerning file access.

What It Does

This system call has two versions — general and specific. The general version returns the total number and LPIDs of all current logical processors. The specific version returns status information of a specific logical processor. You tell AOS/VS which version you want by placing a code in offset ?LPS_PKT.FUNC of the main parameter packet.

If you select the general version, then AOS/VS returns, in a doubleword of the general status subpacket, a bit map that indicates what logical processors exist:

- If a bit is set, then the corresponding LP exists.
- If a bit is not set, then the corresponding LP does not exist.

For example, suppose bits 2, 3, and 5 are set. This means LPs with IDs 2, 3, and 5 exist. Furthermore, LPs with IDs 0, 1, 4, 6, 7, 8, ..., 15 do not exist.

If you select the specific version then AOS/VS returns, in a doubleword of the specific status subpacket, a bit map that indicates what job processors are associated with the given logical processor:

- If a bit is set, then the corresponding JP is associated with the LP.
- If a bit is not set, then no JP is associated with the LP.

For example, suppose bits 2, 3, and 5 are set. This means job processors (JPs) with IDs 2, 3, and 5 are associated with the specified LP. Furthermore, JPs with IDs 0, 1, 4, 6, 7, 8, ..., 15 are not associated with the specified LP.

Figure 2–101 shows the structure of the ?LPSTAT main parameter packet, and Table 2–81 describes its contents. Figure 2–102 and Figure 2–103 show the structure of the general status subpacket and specific status subpacket, respectively. Table 2–82 and Table 2–83 describe the contents of these respective subpackets.

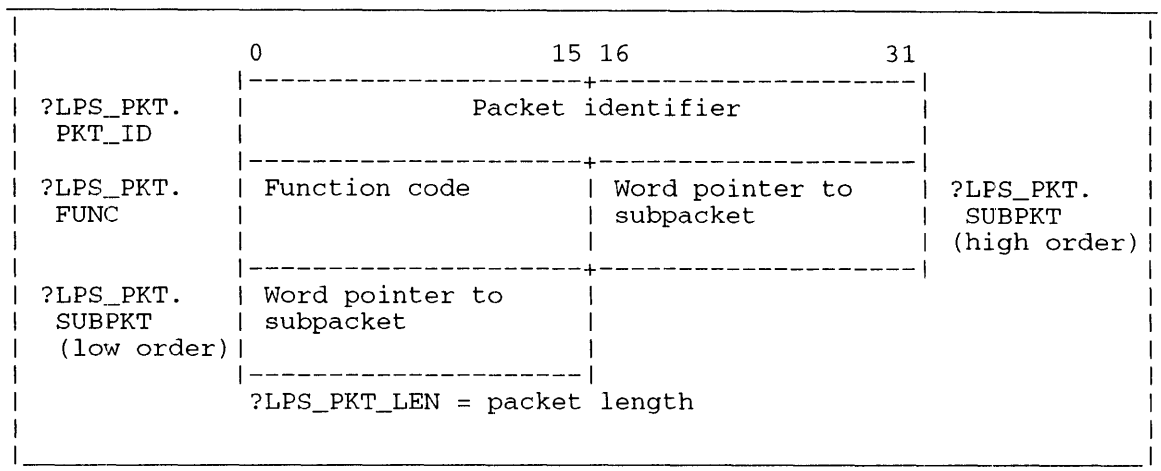


Figure 2–101. Structure of ?LPSTAT Main Packet

Table 2–81. Contents of ?LPSTAT Main Packet

Offset	Contents
?LPS_PKT.PKT_ID (doubleword)	Packet identifier. Place ?LPS_PKT_PKTID here.
?LPS_PKT.FUNC	General/Specific code. Place ?LPS_GEN here to obtain general information or ?LPS_SPEC to obtain specific information. Both values must be between ?LPS_FUNC_MIN and ?LPS_FUNC_MAX, inclusive.
?LPS_PKT.SUBPKT (doubleword)	If you have selected general information, place the word address of the general status subpacket here. If you have selected specific information, place the word address of the specific status subpacket here.

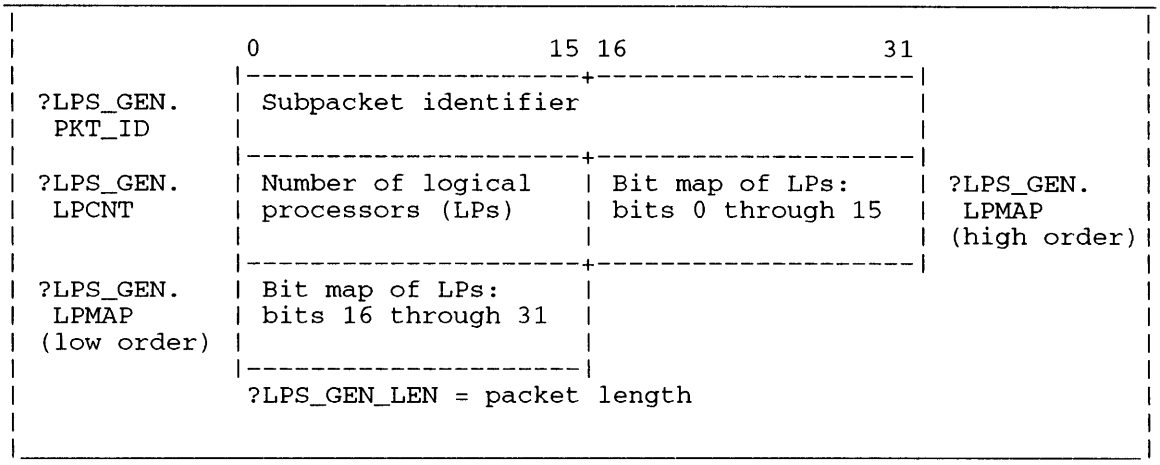


Figure 2–102. Structure of ?LPSTAT General Information Subpacket

Table 2–82. Contents of ?LPSTAT General Information Subpacket

Offset	Contents
?LPS_GEN.PKT_ID (doubleword)	Subpacket identifier. Place ?LPS_GEN_PKTID here.
?LPS_GEN.LPCNT	Number of logical processors that currently exist.
?LPS_GEN.LPMAP (doubleword)	Bit map to indicate the LPIDs of the logical processors that currently exist. For example, if AOS/VS has set the leftmost bit of the first (high-order) word in this doubleword, then logical processor number 0 currently exists.

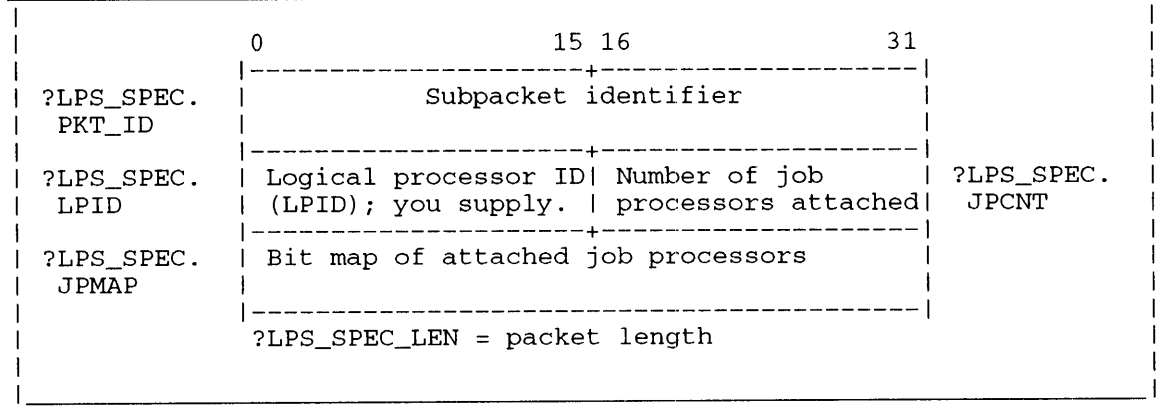


Figure 2–103. Structure of ?LPSTAT Specific Information Subpacket

Table 2–83. Contents of ?LPSTAT Specific Information Subpacket

Offset	Contents
?LPS_SPEC.PKT_ID (doubleword)	Subpacket identifier. Place ?LPS_SPEC_PKTID here.
?LPS_SPEC.LPID	ID number of the logical processor that you want to receive information about.
?LPS_SPEC.JPCNT	Number of job processors attached to the logical processor you specified in the previous word.
?LPS_SPEC.JPMAP (doubleword)	Bit map that indicates the JPIDs of the job processors attached to the logical processor you specified two words ago.

?MAPDV Maps a device into logical address space.

?MAPDV *[packet address]*
error return
normal return

Input

AC0	Reserved (Set to 0.)
AC1	Reserved (Set to 0.)
AC2	Address of the ?MAPDV packet, unless you specify the address as an argument to ?MAPDV

Output

AC0	Unchanged
AC1	Unchanged
AC2	Address of the ?MAPDV packet

Error Codes in AC0

ERPRE	Invalid system call parameter, for any of these reasons – Invalid ?MDPV packet revision (other than zero) – Invalid ?MDRT region type – Invalid ?MDRP starting page (negative) – Invalid ?MDPC number of pages to map
ERMPR	Invalid parameter address (bad packet pointer, etc.)
ERMSI	Map specification illegal for target (not a memory mapped device)
ERNMT	Map target does not exist
ERRAU	Region already in use

Why Use It?

Use this system call to map a region of a process's address space to a special memory mapped device.

Who Can Use It?

The only special privilege needed to issue this call is access devices privilege (value ?PACDEV). There are no restrictions concerning file access.

What It Does

A memory mapped device (?MMAP) is any I/O device that your process can gain access to by reading or writing to a physical page that resolves to the device. A DS/7000–series I/O controller connected to the CPU chassis has a device code associated with it and a node number (determined by the slot the board resides in). DS/7000–series computers include DS/7500–series computers and the ECLIPSE MV/2000 DC computer.

You supply to ?MAPDV a device code, number of pages to map, and logical address to map. ?MAPDV returns the physical address of the first page that it has assigned to that device code. This physical address includes the node associated with the device. You need the node number to gain access to special space to control the device.

?MAPDV performs the functionality of ?STMAP. ?STMAP deals with data channel mapped devices; ?MAPDV deals with memory mapped devices.

Controlling a Memory Mapped I/O Device

The following procedure controls a memory mapped I/O device.

1. Issue ?IDEF and specify a programmable I/O (PIO) device, with no map definition table.
2. Issue ?MAPDV with the following packet offsets:
   - ?MDPK/?MDPL = ?MAPDV_PKT_PKTID.
   - ?MDRT = ?MMAP.
   - ?MDOP = ?MRDO.
   - ?MDIL = 6–bit device code.
   - ?MDRP/?MDRL = 32–bit starting page number (0 = first physical page in the I/O controller).
   - ?MDPC/?MDCL = 32–bit number of pages you need to map.
   - ?MDLA/?MDDL = 32–bit logical address.

   Returned offset ?MDN0/?MDN1 contains the node number needed to gain access to the special space to control the device.

   Returned offset ?MDDT/?MDDL contains the device type (model number) as returned from the SCP command SIZE MEMORY.
3. Do I/O processing as desired.
4. Issue ?MAPDV again with ?UNMR in offset ?MDOP. This is the only offset you have to change.
5. Issue ?IRMV to remove the device specified in offset ?MDDT/?MDDL.

Suppose a process terminates before ?MAPDV, which has ?UNMR in offset ?MDOP, is finished. ?IRMV will ensure that this mapped region is undone. If a process terminates before ?IRMV is finished, then the system will clean up after the termination.

Figure 2–104 shows the structure of ?MAPDV's parameter packet, and Table 2–84 describes its contents.

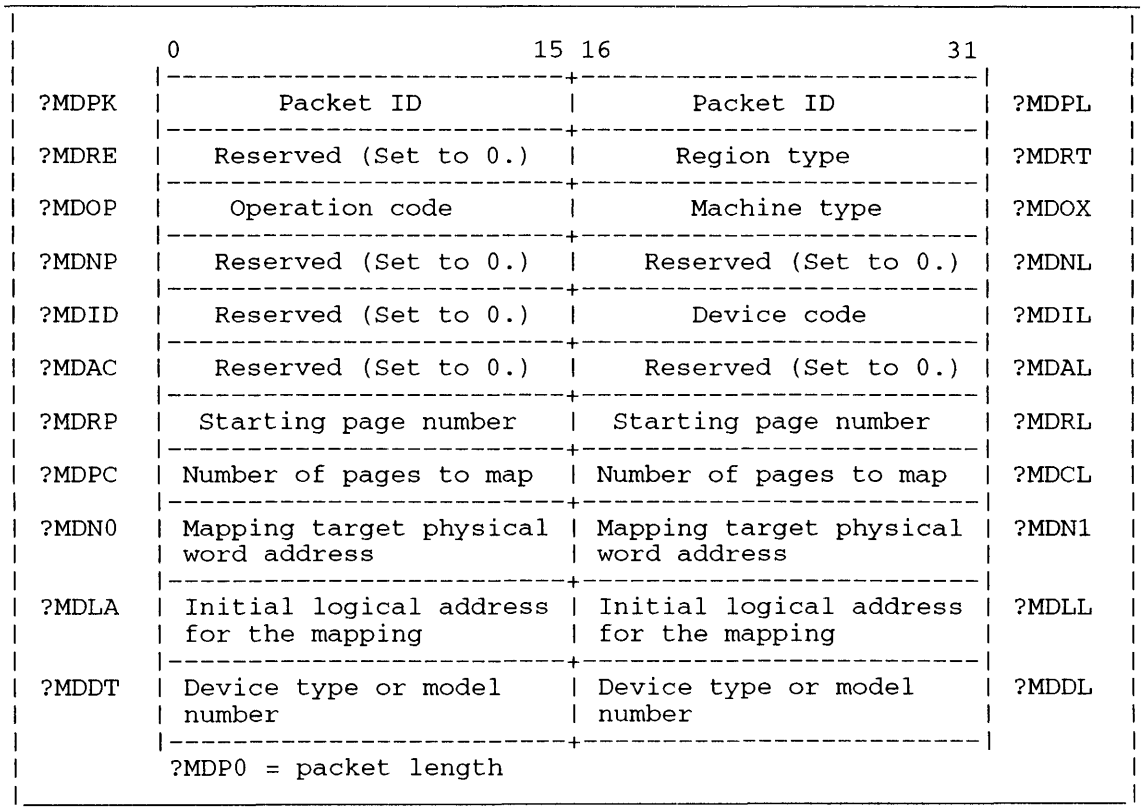


Figure 2–104. Structure of ?MAPDV Packet

Table 2–84. Contents of ?MAPDV Packet

Offset	Contents
?MDPK/ ?MDPL	Packet revision number. Place ?MAPDV_PKT_PKTID here.
?MDRE	Reserved. Set to 0.
?MDRT	Region type. The values and their meanings follow. ?BITM -- Physical video bit map memory. ?VBITM -- Virtual bit map memory. ?MMAP -- Memory mapped device memory.
?MDOP	Operation code. The values and their meanings follow. ?MRDO -- Add region to address space. ?UNMR -- Remove region from address space.
?MDOX	Reserved (AOS/VS and AOS/RT32). AOS/VS II, returns ?MLBS -- LBUS-type machine without data channel maps. ?MMPS -- LBUS-type machine with data channel maps.
?MDNP/ ?MDNL	Reserved. Set to 0.
?MDID	Reserved. Set to 0.
?MDIL	Supply the target device's 6-bit code.
?MDAC/ ?MDAL	Reserved. Set to 0.
?MDRP/ ?MDRL	Supply the starting page number in the target. (0 is the first physical page in the I/O controller.)
?MDPC/ ?MDCL	Supply the number of pages ?MAPDV will map.
?MDN0/ ?MDN1	?MAPDV returns the physical address of the first page that is assigned to the device code of the I/O controller. This address includes the node associated with the target device.
?MDLA/ ?MDLL	Supply the initial logical address for the mapping.
?MDDT/ ?MDDL	?MAPDV returns the device type (model number).

?MBFC — Moves bytes from a customer's buffer.

?MBFC *[packet address]*
error return
normal return

Input

AC0	Reserved (Set to 0.)
AC1	Reserved (Set to 0.)
AC2	Address of the ?MBFC packet, unless you specify the address as an argument to ?MBFC

Output

AC0	Undefined
AC1	Actual number of bytes moved if ?MBFC takes an error return
AC2	Address of the ?MBFC packet

Error Codes in AC0

ERCBK	Connection broken
ERCDE	Connection doesn't exist
ERMPR	System call parameter address error
ERPRV	Caller not privileged for this action (The caller is not a server of the specified customer.)
ERVBP	Invalid byte pointer passed as a system call argument
ERVWP	Invalid word pointer passed as a system call argument

Why Use It?

?MBFC and its companion system call, ?MBTC, give a server process access to a customer's logical address space. These system calls are useful for any application in which the server must move bytes to fulfill a customer's request.

Who Can Use It?

There are no special process privileges needed to issue this call, and there are no restrictions concerning file access. However, the calling process must be a server of the customer process that you specify.

What It Does

?MBFC moves a specific number of bytes from a buffer in a customer's logical address space to a buffer in the server's logical address space.

You can use ?MBFC to move bytes from inner–ring customer buffers, but the address that you specify within the caller's process must be within a ring whose number is greater than or equal to the caller's ring. Also, all source data must reside entirely within the single specified source ring. There must be enough room at the destination for the data to reside entirely within the destination ring specified.

Figure 2–105 shows the structure of the move bytes packet, which both ?MBFC and ?MBTC use. The symbol ?MBLTH represents the length of the packet. Either load the packet address into AC2 before you issue ?MBFC, or cite its address as an argument to ?MBFC.

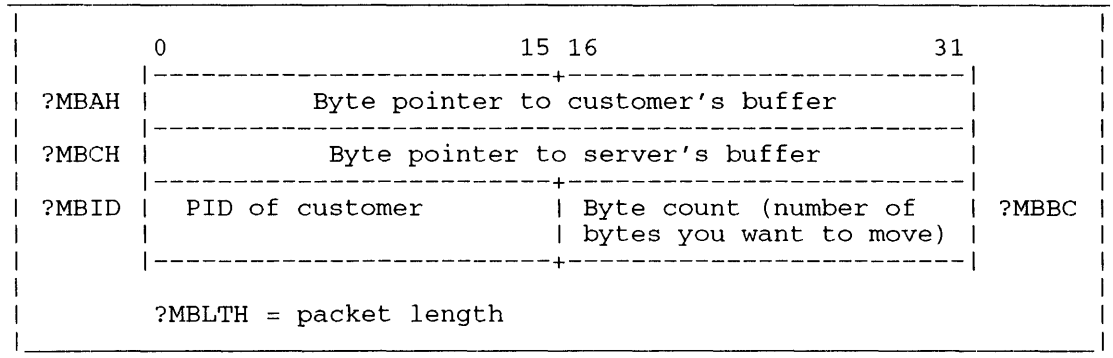


Figure 2–105. Structure of ?MBFC Packet

You can move a maximum of 2048 bytes with ?MBFC. If you specify a byte length longer than 2048, the operating system takes the error return on error code ERMPR, and returns 0 to AC1 (that is, moves 0 bytes). If the server's buffer is too small to accommodate all of the bytes, the operating system moves as many as it can and returns the number of bytes moved to AC1 and error code ERMPR to AC0.

Do not use ?MBFC against a customer that has chained to a new program, unless you can verify the validity of the customer's buffer. Similarly, customer processes should not issue ?CHAIN while they have outstanding requests to servers.

Sample Packet

The following sample packet is set up to move bytes from a customer's buffer:

```
PKT:    .BLK    ?MBLTH          ;Allocate enough space for the
                                ;packet.  Packet length = ?MBLTH.
        .LOC    PKT+?MBAH       ;Byte pointer to customer's buffer.
        .DWORD  CUSBUF*2        ;Customer's buffer is CUSBUF.
        .LOC    PKT+?MBCH       ;Byte pointer to server's buffer.
        .DWORD  SRVBUF*2        ;Server's buffer is SRVBUF.
        .LOC    PKT+?MBID       ;PID of customer.
        .WORD   14.             ;PID of customer is 14.
        .LOC    PKT+?MBBC       ;Number of bytes you want to move.
        .WORD   2048.           ;Move 2048 bytes.  (This is the
                                ;maximum you can transfer.)
        .LOC    PKT+?MBLTH      ;End of packet.
```

Notes

- See the descriptions of ?MBTC and ?CHAIN in this chapter.

?MBTC — Moves bytes to a customer's buffer.

?MBTC *[packet address]*
error return
normal return

Input

AC0	Reserved (Set to 0.)
AC1	Reserved (Set to 0.)
AC2	Address of the ?MBTC packet, unless you specify the address as an argument to ?MBTC

Output

AC0	Undefined
AC1	Actual number of bytes moved if ?MBTC takes an error return
AC2	Address of the ?MBTC packet

Error Codes in AC0

ERCBK	Connection broken
ERCDE	Connection does not exist
ERMPR	System call parameter address error
ERPRV	Caller not privileged for this action (The caller is not a server of the specified customer.)
ERVBP	Invalid byte pointer passed as a system call argument
ERVWP	Invalid word pointer passed as a system call argument

Why Use It?

You can use ?MBTC to move bytes from inner–ring customer buffers.

Who Can Use It?

There are no special process privileges needed to issue this call, and there are no restrictions concerning file access. However, the calling process must be a server of the customer process that you specify.

What It Does

?MBTC moves bytes from a buffer in the server's logical address space to a buffer in the designated customer's logical address space.

You can use ?MBFC to move bytes from inner–ring customer buffers, but the address that you specify within the caller's process must be within a ring whose number is greater than or equal to the caller's ring. Also, all source data must reside entirely within the single specified source ring. There must be enough room at the destination for the data to reside entirely within the destination ring specified.

?MBTC takes the same packet as ?MBFC. (See Figure 2–106.) You can cite the packet address as an argument to ?MBTC, or you can load its address into AC2 before you issue the system call.

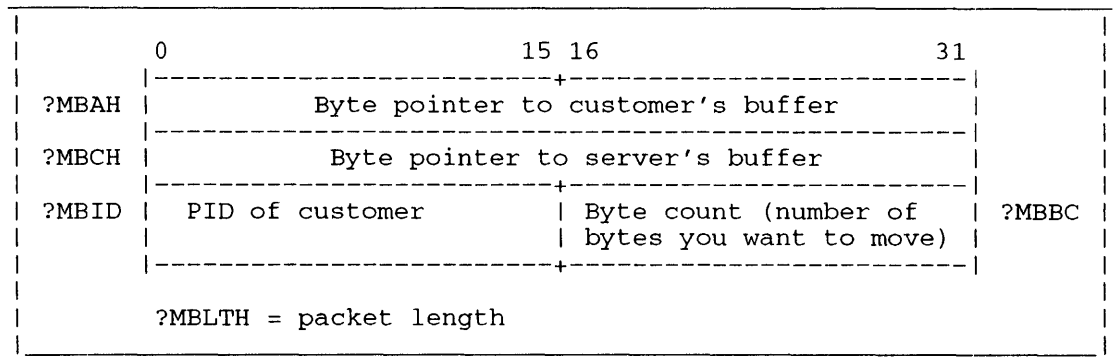


Figure 2–106. Structure of ?MBTC Packet

You can move a maximum of 2048 bytes with ?MBTC. If you specify a larger byte count, ?MBTC fails on error code ERMPR, and the operating system returns 0 to AC1 (0 bytes moved). If the customer's buffer is too small to accommodate the bytes, the operating system moves as many as it can; it returns the number of bytes moved to AC1 and error code ERMPR to AC0.

Do not use ?MBTC against a customer that has chained to a new program, unless you can guarantee the validity of the customer's buffer. (The same restriction applies to ?MBFC.)

Notes

- See the descriptions of ?MBFC and ?CHAIN in this chapter.

?MDUMP — Dumps the memory image from a user–specified ring to a file.

?MDUMP
error return
normal return

Input

AC0	Address that contains ring field of ring to be dumped
AC1	Reserved (Set to 0.)
AC2	One of the following: • Byte pointer to the dump file pathname • –1 to use the default pathname

Output

AC0	Unchanged
AC1	Unchanged
AC2	Unchanged

Error Codes in AC0

ERVBP Invalid byte pointer passed as a system call argument
ERVWP Invalid word pointer passed as a system call argument
File system error codes

Why Use It?

?MDUMP allows you to examine your program memory image when you are not logged on. ?MDUMP complements ?BRKFL, because the dump file contains all of the information in a break file, plus a memory image.

Who Can Use It?

There are no special process privileges needed to issue this call. You must have Read access to the .PR file in the ring that you want to dump. Otherwise, the operating system returns an error.

What It Does

?MDUMP allows you to obtain a snapshot dump of one ring of your address space while your program is executing. The operating system strips the ring field from AC0 and dumps that ring to the dump file that you specify.

The dump file is formatted much like a .PR file in that it has a preamble at the beginning and your actual memory image starts at location 20000. The preamble contains a dump of your task control blocks, which reside within Ring 3 as part of the Agent. You can use the File Editor utility (FED) to examine the dump file. Also, you can run BRAN or the Dump Tool utility on the dump file.

The default name of the dump file has the form

?PID.TIME.RING.MDM

where

PID	is the 5–digit PID of the user.
TIME	is the current time in the form hours_minutes_seconds.
RING	is the ring number of the ring dumped.

An example of such a filename is ?00038.14_27_36.7.MDM.

Notes

- See the description of ?BRKFL in this chapter.

?MEM Lists the current unshared memory parameters.

?MEM
error return
normal return

Input

None

Output

AC0 Maximum number of unshared memory pages currently available

AC1 Number of unshared memory pages currently in use

AC2 Highest unshared address in the caller's logical address space (This value is always "page–adjusted", that is, the OS rounds this value by setting the 10 low–order bits to 1.)

Error Codes in AC0

No error codes are currently defined.

Why Use It?

?MEM is a useful preliminary system call for ?MEMI. Depending on your input parameters, ?MEMI either increases or decreases the number of unshared pages in the logical address space.

Because ?MEM also returns the number of unshared memory pages currently available, you can also use ?MEM to determine your program's unshared size.

Who Can Use It?

There are no special process privileges needed to issue this call, and there are no restrictions concerning file access.

What It Does

?MEM returns the following unshared memory parameters:

- The maximum number of unshared pages available in the caller's logical address space.
- The number of unshared pages currently in use.
- The highest unshared address in the caller's logical address space.

Notes

- See the description of ?MEMI in this chapter.

?MEMI — Changes the number of unshared pages in the logical address space.

?MEMI
error return
normal return

Input

AC0	One of the following: • Number of unshared pages you wish to allocate • Two's complement of the number of unshared pages you wish to remove
AC1	Reserved (Set to 0.)
AC2	Reserved (Set to 0.)

Output

AC0	Unchanged
AC1	Highest logical address in the caller's unshared logical address space
AC2	Undefined

Error Codes in AC0

ERMEM	Insufficient memory available
ERMRL	Memory release error (You tried to remove more pages than the unshared area contains, or you tried to remove Page 0.)

Why Use It?

?MEMI lets you expand the unshared area of the logical address space by using any unused space between the unshared and shared areas. You can also use ?MEMI to reduce the number of pages in the unshared area.

Who Can Use It?

There are no special process privileges needed to issue this call, and there are no restrictions concerning file access.

What It Does

?MEMI increases or decreases the unshared area in the calling process's logical address space by the number of pages that you specify in AC0. The operating system returns the new maximum unshared address to AC1. Note that pages acquired with ?MEMI initially contain all zeros.

To increase the number of unshared pages, load AC0 with a positive value that indicates the number of additional pages you want. To decrease the number of unshared pages, load AC0 with the appropriate two's complement (negative) value.

You can't remove Page 0 of your logical address space.

?MEMI always takes or adds pages at the end of the unshared area. You can't have "holes" in your unshared address space.

Notes

- See the description of ?IREC in this chapter.

?MIRROR — Mirrors and synchronizes LDU images.

AOS/VS

?MIRROR *[packet address]*
error return
normal return

Input

AC0	Reserved (Set to 0.)
AC1	Reserved (Set to 0.)
AC2	Address of the ?MIRROR packet, unless you specify the address as an argument to ?MIRROR.

Output

AC0	Unchanged
AC1	Unchanged
AC2	Address of the ?MIRROR packet

Error Codes in AC0

ERCNM Controller does not support logical disk unit (LDU) mirroring
ERDMO Disk marked as owned by another system
ERDRS Device reserved by another port
ERIFT Illegal file type
ER_FS_INVALID_MIRROR_FUNCTION
Invalid mirror function
ER_FS_INVALID_MIRROR_OPTIONS
Invalid mirror options
ERLDR LDU released during synchronization
ERLDX LDU does not exist
ERLFM LDU format mismatch — not a valid LDU mirror
ERLMM LDU name mismatch — not a valid LDU mirror
ERLSZ LDU size mismatch — not a valid LDU mirror
ER_FS_LDM_MAX_NUM_IMAGES_CURRENTLY_ESTABLISHED
Maximum number of LDU images exceeded
ERMOP Mirrored LDU is out of phase
ER_FS_OWNER_ACCESS_REQ
Owner access required
ERRID LDU IDs are not unique — not a valid LDU mirror
ERSFL Mirrored LDU synchronization failed
ERTMM Controller cannot support an additional LDU mirror

Why Use It?

You use ?MIRROR to manage your mirrored LDU (logical disk unit) configuration. The call controls all changes in the state of mirroring.

Who Can Use It?

There are no special process privileges needed to issue this call. You must have Owner access to the target LD's root and Execute access to the physical disks specified.

What It Does

?MIRROR allows you to add an image in a mirrored LDU configuration. LDU mirroring allows you to maintain two images of the same LDU. If one LDU image becomes unavailable, you still have access to the LDU data on the remaining available image.

Adding an image to a mirrored LDU requires that ?MIRROR synchronize the images so that they contain the same data. When ?MIRROR performs this synchronization, you direct it to either wait for the synchronization to complete or just invoke the synchronization.

Issuing ?MIRROR might use significant system resources when you synchronize a mirror. The related system task will use appreciable I/O resources and memory/CPU resources.

?MIRROR operates on LDU images. You frequently issue ?MIRROR along with ?XINIT and ?RELEASE since these two system calls also operate on LDU images. ?RELEASE terminates ?MIRROR if you issue either the 32-bit version of ?MIRROR with bit ?MIRROR_SYNC_WAIT set in offset ?MIRROR_PKT.OPTIONS or else the 16-bit version with bit ?MBWAIT set in offset ?MIOP.

Figure 2-107 shows the structure of the ?MIRROR main packet, and Table 2-85 describes its contents. Figure 2-108 shows the structure and contents of the ?MIRROR subpacket. This subpacket is identical to the one that ?XINIT uses except for the value of the packet identifier.

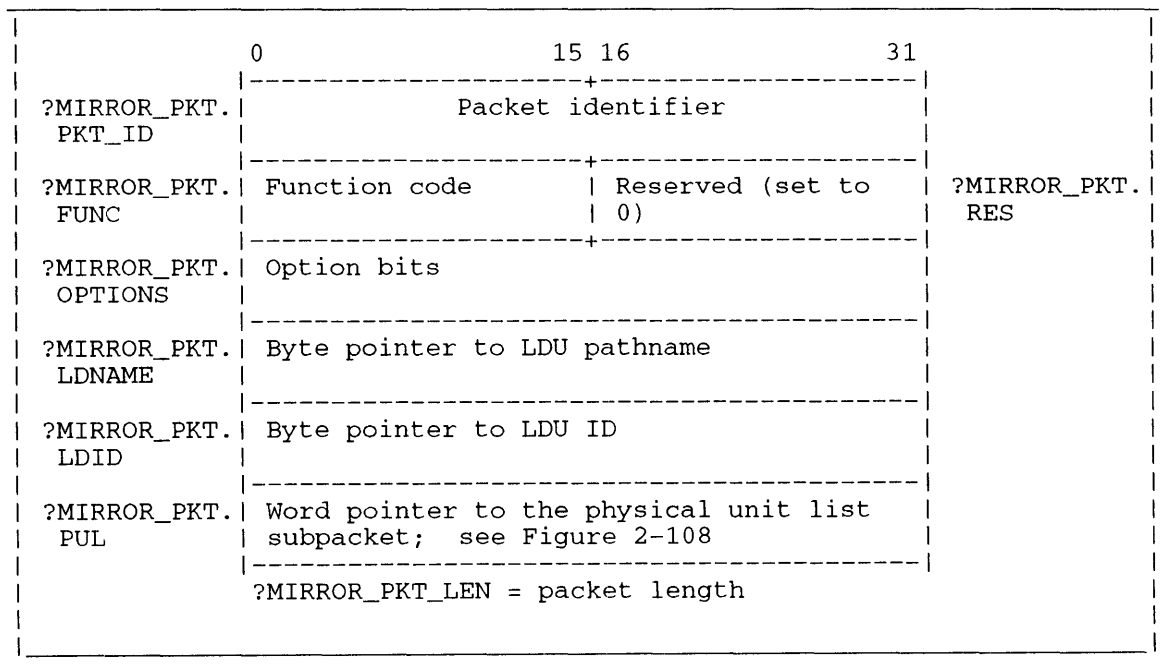


Figure 2-107. Structure of ?MIRROR Packet

Table 2–85. Contents of ?MIRROR Packet

Offset	Contents
?MIRROR_PKT.PKT_ID (doubleword)	Packet identifier. Place ?MIRROR_PKT_PKTID here.
?MIRROR_PKT.FUNC	Function code. Place one of the following values here: ?MIRROR_SYNCHRONIZE — to synchronize the mirror (i.e., add an image to the mirrored set.) ?MIRROR_BREAK — to remove an image from the mirrored set.
?MIRROR_PKT.RES	Reserved. (Set to 0.)
?MIRROR_PKT.OPTIONS (doubleword)	Options word. Choose from the following values here, but choose ?MIRROR_NO_HARDWARE and ?MIRROR_OVERRIDE and ?MIRROR_SYNC_WAIT only when offset ?MIRROR_PKT.FUNC contains ?MIRROR_SYNCHRONIZE: ?MIRROR_LDID_SPECIFIED — If the physical disk contains more than one logical disk piece, then you must set this bit to indicate that you have specified at least one LDUID. Also, you must specify the LDUID in offset ?MIRROR_PKT.LDID. This value applies only to the new file system. ?MIRROR_NO_HARDWARE — to prevent the system from mirroring the LDU in hardware; hardware mirroring is the default case whenever possible. This value applies only to the new file system. ?MIRROR_TRESPASS — to trespass on a device marked as owned by another system or to trespass on a device that another port has reserved. This value applies only to the new file system. ?MIRROR_OVERRIDE — to override the error condition ERMOP. ("Mirrored LDU is out of phase.") ?MIRROR_SYNC_WAIT — to wait for the synchronization process to complete. (The system might return error code ERSFL.)

(continued)

Table 2–85. Contents of ?MIRROR Packet

Offset	Contents
?MIRROR_PKT.LDNAME (doubleword)	Supply a byte pointer to the pathname of the LDU name that the ?MIRROR call addresses. If any physical disk contains more than one logical disk piece, then set bit ?MIRROR_LDID_SPECIFIED in offset ?MIRROR_PKT.OPTIONS and supply a byte pointer to the LD image name in offset ?MIRROR_PKT.LDID. If the specified LDU image is not associated with the specified LDU name, the system returns an error.
?MIRROR_PKT.LDID (doubleword)	If you have set bit ?MIRROR_LDID_SPECIFIED in offset ?MIRROR_PKT.OPTIONS, then supply a null-terminated byte pointer to a valid LDU ID. If you have not set this bit, then supply 0. Always supply 0 for the old file system.
?MIRROR_PKT.PUL (doubleword)	Supply a word pointer to the physical unit list subpacket. Figure 2-108 describes this subpacket.

(concluded)

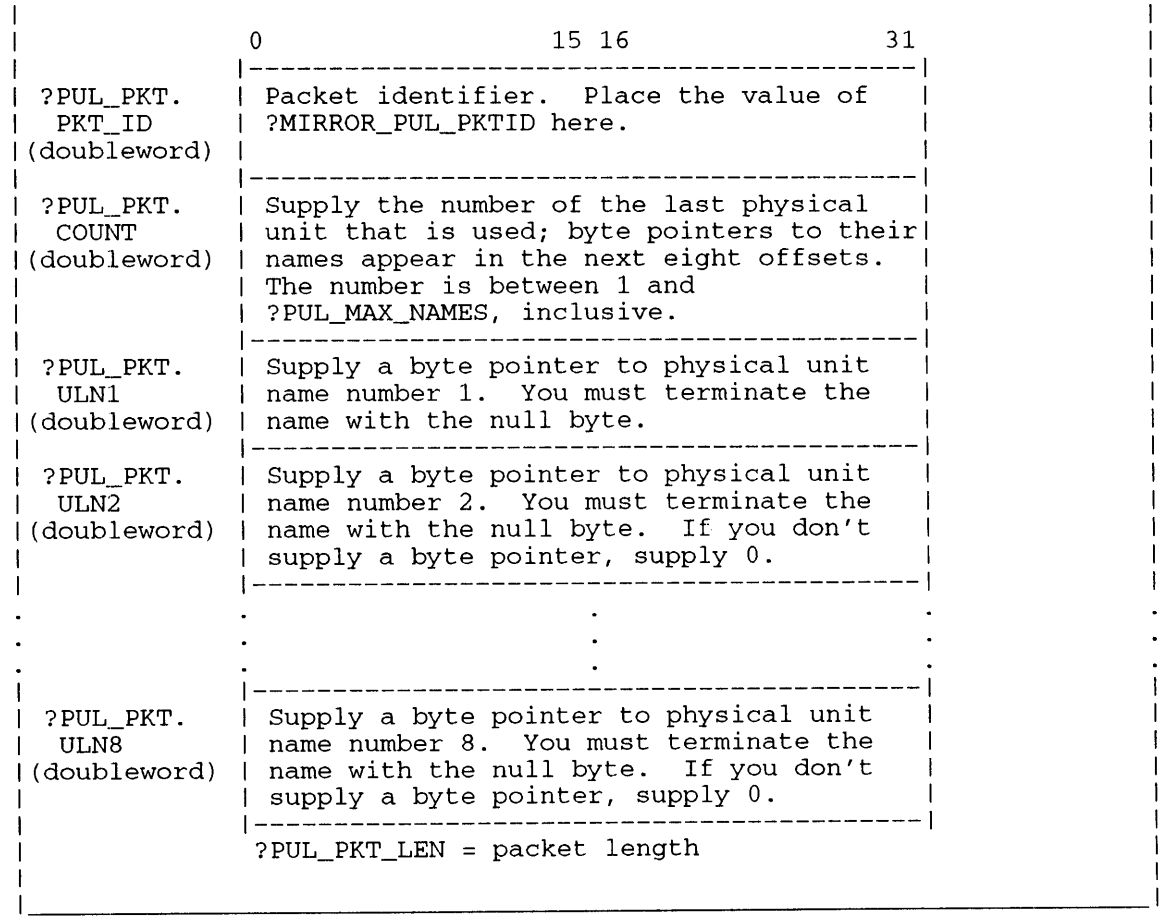


Figure 2–108. Structure of ?MIRROR Subpacket

16–bit Version of ?MIRROR System Call

A 16–bit version of this system call also exists. Its functionality is identical to that of the 32–bit version previously described.

Figure 2–109 contains the structure of the 16–bit ?MIRROR call main packet and Figure 2–110 contains the structure of the 16–bit ?MIRROR call subpacket. The definitions of the offsets in these packets are identical to those in Figure 2–107 and Figure 2–108, respectively. The only differences in the corresponding packets are the names of the offsets.

Table 2–86 describes the contents of Figure 2–109.

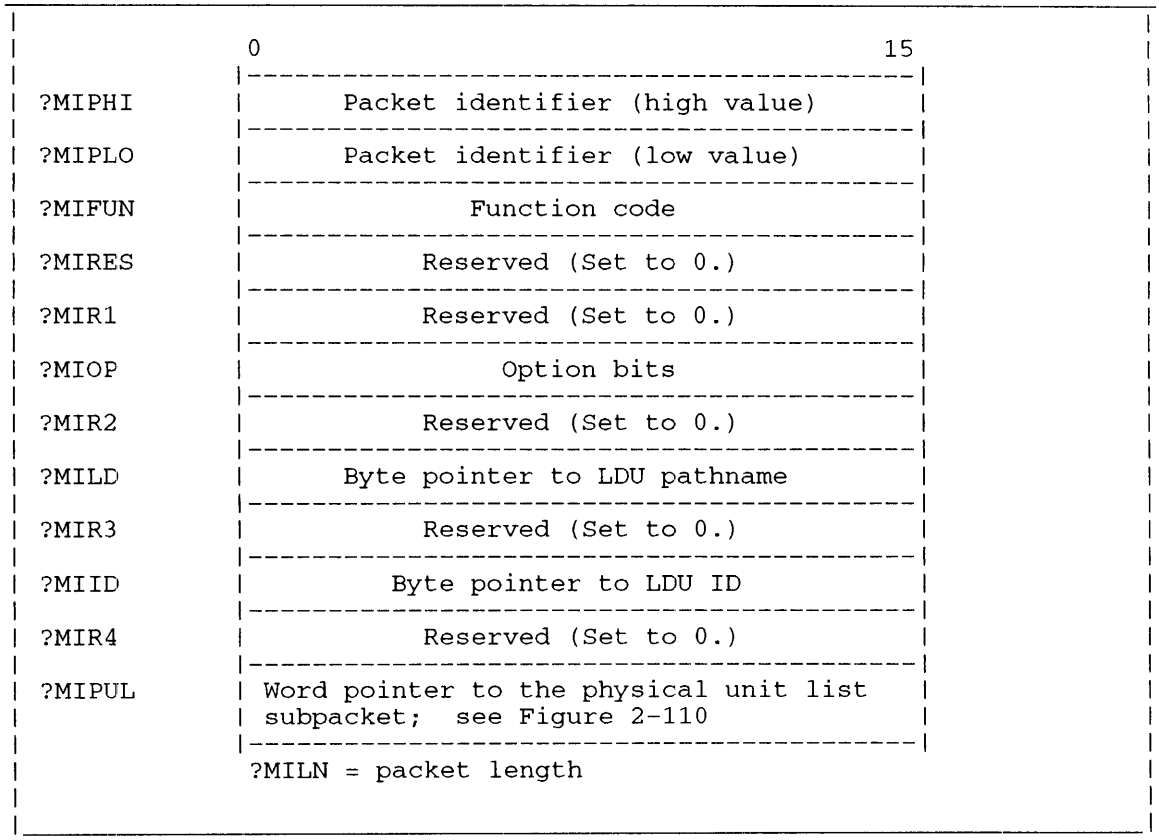


Figure 2–109. Structure of 16–Bit ?MIRROR Packet

Notes

- See the descriptions of ?RELEASE and ?XINIT in this chapter.

Table 2–86. Contents of 16–Bit ?MIRROR Packet

Offset	Contents
?MIPHI	Packet identifier (high word). Place ?MII1 here.
?MIPLO	Packet identifier (low word). Place ?MII2 here.
?MIFUN	Function code. Place one of the following values here: ?MFSYM — to synchronize the mirror (i.e., add an image to the mirrored set.) ?MFBRK — to remove an image from the mirrored set.
?MIRES	Reserved. (Set to 0.)
?MIR1	Reserved. (Set to 0.)
?MIOP	Options word. Choose from the following values here, but choose ?MBNHR and ?MBOOP and ?MBWAIT only when offset ?MIFUN contains ?MFSYM. ?MBNHR — to prevent the system from mirroring the LDU in hardware; hardware mirroring is the default case whenever possible. This value applies only to the new file system. ?MBNLD — If the physical disk contains more than one logical disk piece, then you must set this bit to indicate that you have specified at least one LDUID. Also, you must specify the LDUID in offset ?MIID. This value applies only to the new file system. ?MBTRP — to trespass on a device marked as owned by another system or to trespass on a device that another port has reserved. This value applies only to the new file system. ?MBOOP — to override the error condition ERMOP. ("Mirrored LDU is out of phase.") ?MBWAIT — to wait for the synchronization process to complete. (The system might return error code ERSFL.)

(continued)

Table 2–86. Contents of 16–Bit ?MIRROR Packet

Offset	Contents
?MIR2	Reserved. (Set to 0.)
?MILD	Supply a byte pointer to the pathname of the LDU name that the ?MIRROR call addresses. If any physical disk contains more than one logical disk piece, then set bit ?MBNLD in offset ?MIOP and supply a byte pointer to the LD image name in offset ?MIID. If the specified LDU image is not associated with the specified LDU name, the system returns an error.
?MIR3	Reserved. (Set to 0.)
?MIID	If you have set bit ?MBNLD in offset ?MIOP, then supply a null-terminated byte pointer to a valid LDU ID. If you have not set this bit, then supply 0. Always supply 0 for the old file system.
?MIR4	Reserved. (Set to 0.)
?MIPUL	Supply a word pointer to the physical unit list subpacket. Figure 2-110 describes this subpacket.

(concluded)

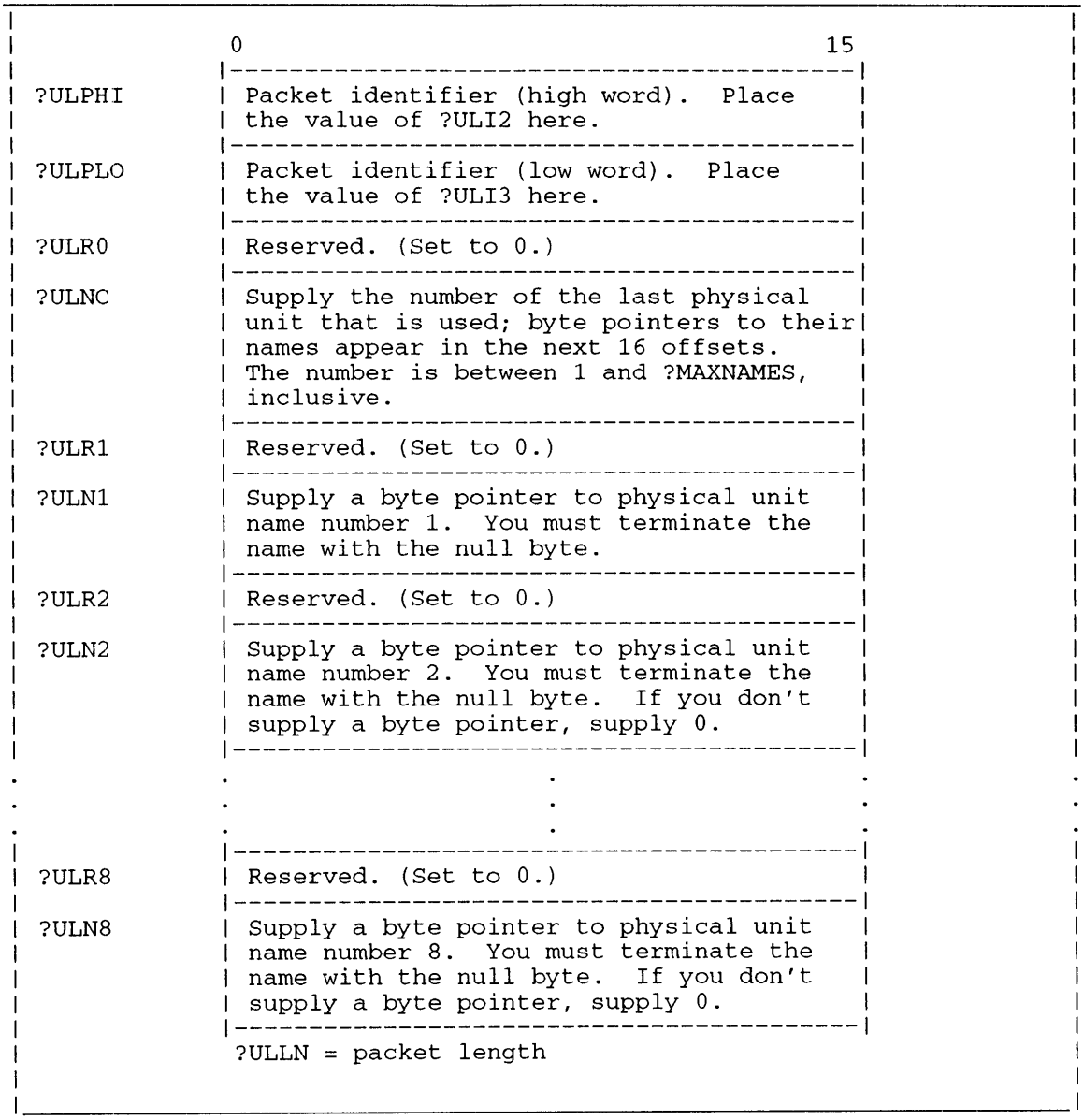


Figure 2–110. Structure of 16–Bit ?MIRROR Subpacket

?MPHIST — Starts a histogram on a uni– or multi–processor system (32–bit processes only).

AOS/VS

?MPHIST *[packet address]*
error return
normal return

Input

AC0	Reserved (Set to 0.)
AC1	Reserved (Set to 0.)
AC2	Address of the ?MPHIST packet, unless you specify the address as an argument to ?MPHIST

Output

AC0	Unchanged
AC1	Unchanged
AC2	Address of the ?MPHIST packet

Error Codes in AC0

ERHIS	Error on histogram initialization or deletion
ERMPR	System call parameter address error
ERPNM	Illegal process name
ERPRH	Attempt to access process not in hierarchy
ERPRV	Caller not privileged for this action (The caller is not a resident process.)
ERVBP	Invalid byte pointer passed as a system call argument
ERVWP	Invalid word pointer passed as a system call argument

Why Use It?

Use this system call to start a histogram on either a uniprocessor system or on a multiprocessor system.

Who Can Use It?

The caller must be a resident process. Furthermore, you must have Superprocess privilege if you start a histogram on a process that is not in your hierarchy. There are no restrictions concerning file access.

What It Does

This system call starts a histogram for a range of local addresses in a process. The caller cannot activate more than one histogram at a time.

If you want information about the calling process, place the value of ?MPH_PID in offset ?MPH_PKT.CODE of the packet and place –1 in offset ?MPH_PKT.PID.

If you want information about another process, place the value of ?MPH_PNAME in offset ?MPH_PKT.PCODE. Either supply the process ID in offset ?MPH_PKT.PID (and 0s in offsets ?MPH_PKT.PNAME and ?MPH_PKT.PSUPPLIED), or supply 0 in offset ?MPH_PKT.PID and the appropriate byte pointer information in offsets ?MPH_PKT.PNAME and ?MPH_PKT.PSUPPLIED.

If you want information about all processes, place the value of ?MPH_PID in offset ?MPH_PKT.PCODE of the packet and place 0 in offset ?MPH_PKT.PID.

Supply the rest of the histogram packet, which includes the starting and ending addresses of the range that ?MPHIST monitors, the size of each interval in the specified range, and the address of the receive buffer for each job processor. If you don't want to specify a receive buffer for a job processor, set this address offset (?MPH_PKT.JP0, ..., ?MPH_PKT.JP15) to 0.

Note that the format of the receive buffer is the same for ?WHIST and ?MPHIST. However, ?MPHIST can have a receive buffer for each job processor.

Setting the interval size to 0 results in a simple histogram that records how often the target process gained CPU control. AOS/VS ignores the starting and ending addresses.

AOS/VS returns histogram statistics in array format to each of your specified receive buffers. The array consists of a fixed–length header followed by double–word entries that correspond to each specified interval.

Side Effects

You can specify receive buffers for job processors that have not been initialized. Why should you? When a job processor is initialized, it can start writing numbers to its receive buffer dynamically. AOS/VS places zeros in the receive buffers at the start of the call regardless of whether the processor has been initialized.

Also, you don't have to specify a receive buffer for every job processor. This is useful if you want a histogram that looks at activity on one job processor only. It will cut down on the amount of space your program uses since you don't have to set aside multiple copies of your receive buffer.

The flexibility of ?MPHIST is such that you decide on the interpretation of the returned data. If you've allocated receive buffers for all job processors and they are homogeneous, then you should be able to average the numbers from all the receive buffers. Such an average yields histogram data for the process on the system. However, if you haven't allocated receive buffers for all the job processors or if not all the job processors are homogeneous, then averaging the numbers gives skewed results. In such a case it would be useful to look at the data on a job processor basis.

Figure 2–111 shows the structure of the ?MPHIST parameter packet, and Table 2–87 describes its contents. Table 2–88 shows the structure and contents of a buffer that receives histogram statistics. See the explanation of ?GHRZ for details about the system clock's frequency.

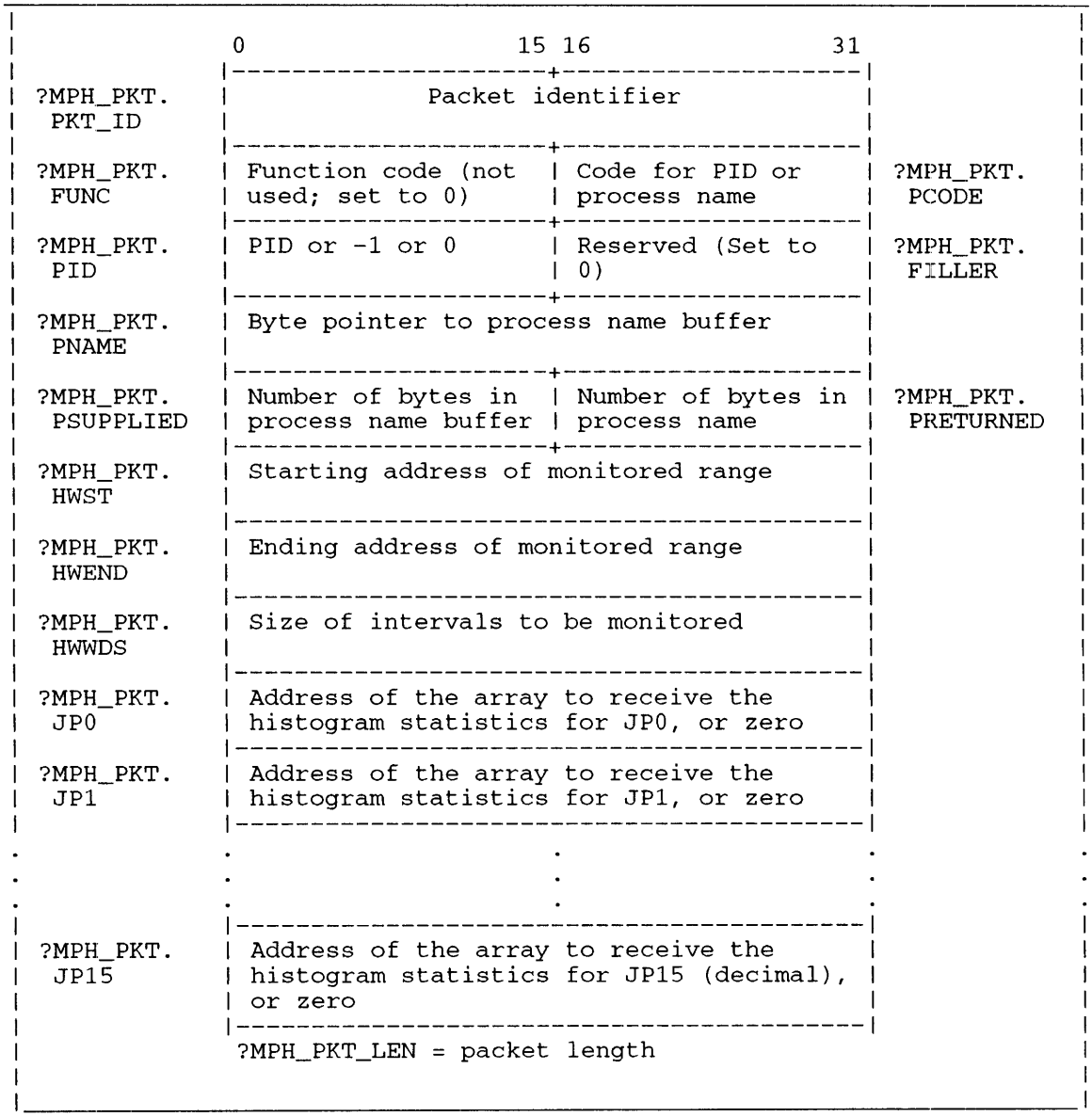


Figure 2–111. Structure of ?MPHIST Packet

Table 2–87. Contents of ?MPHIST Packet

Offset	Contents
?MPH_PKT.PKT_ID (doubleword)	Packet identifier. Place ?MPHIST_PKT_PKTID here.
?MPH_PKT.FUNC	Function code. Not used. (Set to 0.)
?MPH_PKT.PCODE	Code word into which you place ?MPH_PID when you supply a PID or else ?MPH_PNAME when you supply a process name.
?MPH_PKT.PID	Supply -1 for the calling process. For another process, either place its PID here or else place place 0 here and the process's name information in the two offsets after the next one. For all processes, place 0 here and in the two offsets after the next one.
?MPH_PKT.FILLER	Reserved. (Set to 0.)
?MPH_PKT.PNAME (doubleword)	If you've placed 0 in offset ?MPH_PKT.PID, then place a byte pointer here pointing to the process's name and place its length in the next offset. Or, place a 0 here for information about all processes.
?MPH_PKT.PSUPPLIED	If you've placed 0 in offset ?MPH_PKT.PID and a byte pointer in the previous offset, then place the number of bytes in the byte pointer buffer here. Or, place a 0 here for information about all processes.
?MPH_PKT.PRETURNED	If you've specified a byte pointer in offset ?MPH_PKT.PNAME, then AOS/VS returns the actual number of bytes in the string in the byte pointer buffer. If you haven't specified a byte pointer in offset ?MPH_PKT.PNAME, place 0 here.
?MPH_PKT.HWST	Supply the starting address of the memory area that you want monitored.
?MPH_PKT.HWEND	Supply the ending address of the memory area that you want monitored.
?MPH_PKT.HWWDS	Supply the number of intervals in the memory area that you want monitored.

(continued)

Table 2–87. Contents of ?MPHIST Packet

Offset	Contents
?MPH_PKT.JP0	Place zero here if you don't want to receive histogram statistics for job processor 0, or else place the word address here of the array that will receive the histogram statistics.
?MPH_PKT.JP1	Place zero here if you don't want to receive histogram statistics for job processor 1, or else place the word address here of the array that will receive the histogram statistics.
.	.
.	.
.	.
?MPH_PKT.JP15	Place zero here if you don't want to receive histogram statistics for job processor 15, or else place the word address here of the array that will receive the histogram statistics.

(concluded)

Table 2–88. Structure and Contents of ?MPHIST Histogram Array

	0 15 16 31
?MPHIST_ HTTH	Total number of real-time clock pulses (ticks) counted in this histogram
?MPHIST_ HPRH	Total number of ticks when program counter (PC) was within the target process, but outside the specified range
?MPHIST_ HAPH	Total number of ticks in other processes
?MPHIST_ HSBH	Total number of ticks in AOS/VS, except those recorded when it was in an idle loop
?MPHIST_ HSIH	Total number of ticks in a system idle loop
?MPHIST_ HARAY	Total number of ticks in the first interval
?MPHIST_ HARAY + 2	Total number of ticks in the second interval
.	.
.	.
.	.
?MPHIST_ HARAY + 2*(n-1)	Total number of ticks in the nth interval

?MYTID — Gets the priority and TID of the calling task.

?MYTID
error return
normal return

Input

None

Output

AC0	TID of the calling task
AC1	Priority level of the calling task
AC2	Undefined

Error Codes in AC0

No error codes are currently defined.

Why Use It?

?MYTID allows you to identify the calling task by its priority and TID. ?MYTID can be a useful preliminary to other tasking system calls that require TIDs and/or priority levels as input parameters.

Who Can Use It?

There are no special process privileges needed to issue this call, and there are no restrictions concerning file access.

What It Does

?MYTID returns the TID of the calling task to AC0 and its priority level to AC1.

AOS/VS

?NTIME *[packet address]*
error return
normal return

Input

AC0	Reserved (Set to 0.)
AC1	Reserved (Set to 0.)
AC2	Address of the ?NTIME packet, unless you specify the address as an argument to ?NTIME

Output

AC0	Unchanged
AC1	Unchanged
AC2	Address of the ?NTIME packet

Error Codes in AC0

ERICD	Illegal function code
ERPRV	Caller not privileged for this action (It is not PID 2 and it does not have System Manager privilege turned on.)
ERPVS	Packet version not supported (bad value in ?TIME_PKT.PKT_ID)
ERRVN	Reserved value not zero
ERTIM	Input time (or date) argument is out of range
ERVWP	Invalid word pointer passed as system call argument

Why Use It?

The system manager can use this call to change the time, date, and time zone. This change would occur because of going on or off Daylight Savings Time near the beginning and end of summer. After the change, system call ?GTIME returns information based on the values given to ?NTIME.

Who Can Use It?

Only the system operator (PID 2) or a process with System Manager privilege can issue ?NTIME.

What It Does

This system call sets the current time, date, and time zone. Time and date are typically the date and time at the site with the ECLIPSE MV/Family hardware on which the operating system is running.

Time zones are based on Universal Time (UTC). This is the international standard time derived from solar time at the meridian passing through Greenwich, England (the prime meridian, longitude 0 degrees). An older name for Universal Time is Greenwich Mean Time (GMT).

Time zones decrement as a person moves west of Greenwich, England; they increment as the person moves east of this city. For example, Mountain Standard Time is 7 hours west of

Greenwich. Its time code is –7:00, so you could use ?NTIME to place –7 and 0 respectively in offsets ?TIME_PKT.ZONE_HOUR and ?TIME_PKT.ZONE_MINUTE of the parameter packet. An operator in Phoenix, Arizona (United States of America) might provide this information. For another example, Calcutta, India is in a time zone +5–1/2 hours east of Greenwich. You could use ?NTIME to place 5 and 30. respectively in offsets ?TIME_PKT.ZONE_HOUR and ?TIME_PKT.ZONE_MINUTE of the parameter packet.

You can supply the time and date in local time or in Universal Time. To supply the data in local time, place ?TIME_PKT_LOCAL in offset ?TIME_PKT.FUNC. To supply the data in Universal Time, place ?TIME_PKT_UTC in the offset. Issuing ?NTIME updates auxiliary clocks such as the SCP boot clock.

?NTIME is the inverse of ?GTIME. You can issue ?GTIME, update its parameter packet as desired, and then issue ?NTIME with the same parameter packet.

The time data you enter with ?NTIME must be such that both the local time and Universal Time are between two values. The lower value is midnight, January 1, 1968 (1968–01–01:00:00:00). The upper value is 1 second before the 22nd century begins (2099–12–31:23:59:59). Also, the two time zone offsets must between –12:00 and +13:00 inclusive.

16–bit programs should not issue ?NTIME because the packet for ?NTIME does not appear in PARU.16.SR.

Figure 2–112 shows the structure of the ?NTIME parameter packet, and Table 2–89 describes its contents.

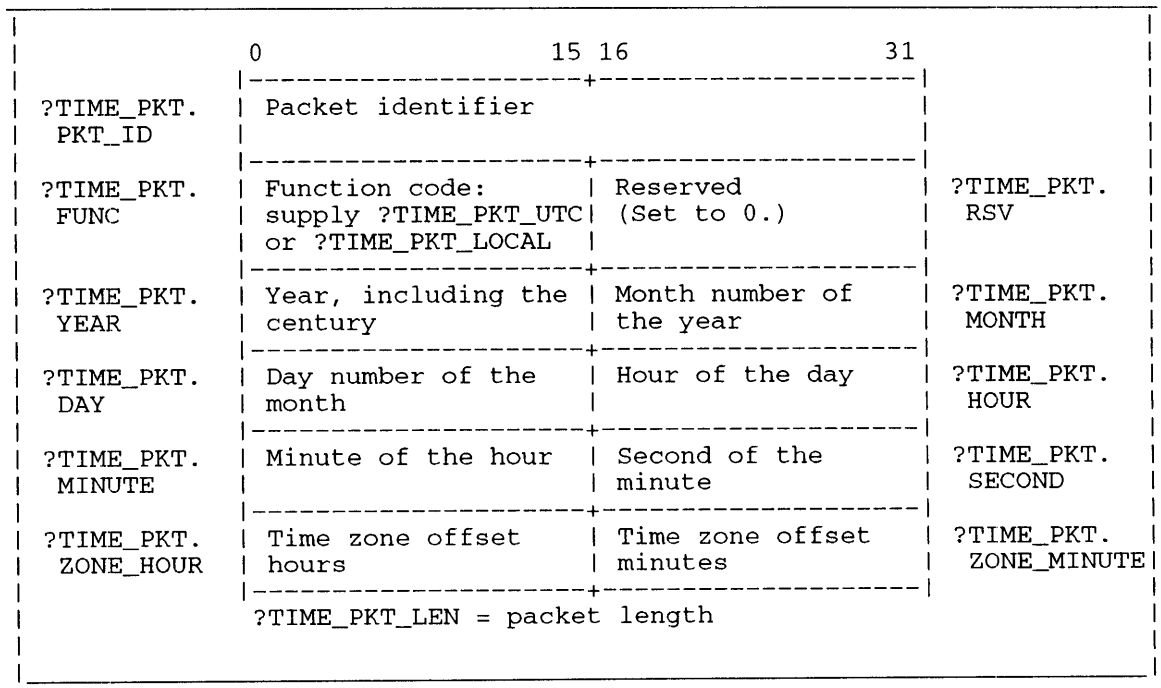


Figure 2–112. Structure of ?NTIME Packet

Table 2–89. Contents of ?NTIME Packet

Offset	Contents
?TIME_PKT.PKT_ID (doubleword)	Packet identifier. Place ?TIME_PKT_PKTID here.
?TIME_PKT.FUNC	Function code. To supply Universal Time, place ?TIME_PKT_UTC here; to supply local time, place ?TIME_PKT_LOCAL here.
?TIME_PKT.RSV	Reserved. (Set to 0.)
?TIME_PKT.YEAR	Specify the current year as a number greater than 1967.
?TIME_PKT.MONTH	Specify the current month as a number from 1 to 12 (1 for January, 2 for February, ..., 12 for December).
?TIME_PKT.DAY	Specify the current day number of the month (1 for the 1st, 2 for the 2nd, ..., 31 for the 31st).
?TIME_PKT.HOUR	Specify the current hour number of the day as a number from 0 to 23, inclusive (i.e., a 24-hour clock).
?TIME_PKT.MINUTE	Specify the current minute number of the hour as a number from 0 to 59, inclusive.
?TIME_PKT.SECOND	Specify the current second number of the minute as a number from 0 to 59, inclusive.
?TIME_PKT. ZONE_HOUR	Specify the current offset in hours relative to Universal Time. Western Hemisphere zones have negative values, Eastern Hemisphere zones have positive values, and the zone whose center is the prime meridian has a zero value. For example, West Germany is one time zone east of the prime meridian; specify 1 in this word for the 1 hour offset (i.e., difference).
?TIME_PKT. ZONE_MINUTE	Specify the current offset in minutes relative to ?TIME_PKT.ZONE_HOUR. For most countries this value is zero, but for a few -- such as India -- this value is nonzero. India is five time zones east of the prime meridian. When it is 0300 (3:00 am) in Greenwich it is 0830 (8:30 am) in India; consequently supply 30 (decimal) to ?TIME_PKT.ZONE_MINUTE and supply 5 to ?TIME_PKT.ZONE_HOUR.

Notes

- See the descriptions of ?GTIME and ?RTODC in this chapter.

?ODIS
error return
normal return

Input

None

Output

None

Error Codes in AC0

No error codes are currently defined.

Why Use It?

By default, the operating system enables terminal interrupts when each program begins to execute. ?ODIS lets you override this default or lets you revoke an interrupt enable that was caused by an ?OEBL, ?INTWT, or ?CHAIN.

Who Can Use It?

There are no special process privileges needed to issue this call, and there are no restrictions concerning file access.

What It Does

?ODIS disables all terminal interrupts (Ctrl–C Ctrl–A sequences) until one of the following occurs:

- You explicitly re–enable Ctrl–C Ctrl–A sequences by issuing ?OEBL.
- You issue ?INTWT to define a terminal interrupt task (a task to monitor the process terminal for Ctrl–C Ctrl–A sequences).
- You issue ?CHAIN to chain to a new program.

Notes

- See the descriptions of ?OEBL, ?INTWT, and ?CHAIN in this chapter.

?OEBL
error return
normal return

Input

None

Output

None

Error Codes in AC0

No error codes are currently defined.

Why Use It?

You can use ?OEBL to revoke a previous ?ODIS or to re–enable terminal interrupts after an ?INTWT sequence.

Who Can Use It?

There are no special process privileges needed to issue this call, and there are no restrictions concerning file access.

What It Does

?OEBL is the inverse of ?ODIS; that is, it re–enables terminal interrupts (Ctrl–C Ctrl–A sequences). Note that ?OEBL does not define a task for processing terminal interrupts. (To do this, issue ?INTWT.)

Notes

- See the descriptions of ?ODIS, ?INTWT, and ?CHAIN in this chapter.

?OPEN — Opens a file.

?OPEN *[packet address]*
error return
normal return

Input

AC0	Reserved (Set to 0.)
AC1	Reserved (Set to 0.)
AC2	Address of the ?OPEN packet, unless you specify the address as an argument to ?OPEN

Output

AC0	Undefined
AC1	Undefined
AC2	Address of the ?OPEN packet

Error Codes in AC0

EREO1	File is open, can't exclusively open
EREO2	File exclusively opened, can't open
ERFAD	File access denied
ERIFL	IAC (Intelligent Asynchronous Controller) failure
ERINP	IPC file not opened by another proc
ERIPS	Illegal pipe size
ERMPR	System call parameter address error
ERP00	Illegal pipe open option
ERVWP	Invalid word pointer passed as a system call argument

Who Can Use It?

There are no special process privileges needed to issue this call. You need Execute access to the file's directory. You also need Write access to the file's directory if issuing ?OPEN will create the file. And, you need Read access to the file and its directory if you are issuing ?OPEN against a file that presently exists.

What It Does

?OPEN opens a file or device for I/O and directs the operating system to assign it a unique channel number. A single process can open as many as 224 channels at the same time. Note that you cannot use ?OPEN to open a file for block I/O, physical block I/O, shared access, or to open a binary synchronous communications line.

You can use ?OPEN to create and then open a file. If you choose this creation option and you choose the default file–type specification, the operating system creates the file as a user data file. (User data files are not executable program files.)

The packets for ?OPEN, ?READ, ?WRITE, and ?CLOSE have the same structure, although not all offsets apply to every system call. You can specify the packet address as an argument to ?OPEN or load AC2 with the packet address before you issue ?OPEN. In both cases, the operating system returns the packet address in AC2.

?OPEN Continued

Figure 2–113 shows the structure of the ?OPEN packet, and Table 2–90 describes each offset and bit position in the packet. The accompanying text explains the options you can exercise in setting the packet specifications.

Figure 2–113 includes byte pointers to extensions for the basic I/O packet. These extensions direct the operating system to perform extended processing on the target file. See the following section "Packet Versions."

The following guidelines apply to the ?OPEN packet:

- You can default some specifications, such as record format, and use the values you set for them when you created the file. You can also alter some of the ?OPEN specifications, such as record format and buffer address, when you read to or write from the file.
- When you close and reopen a file, the operating system overwrites the default values in the packet. Thus, you must reset these values before you use the packet again.
- The parameters in the file specifications word (?ISTI) represent bit masks, not individual bits. To select more than one option for ?ISTI, OR the appropriate masks. For example, the ?ISTI specification ?IEXO!?OFIN opens the file for the exclusive use of the calling task for read purposes only.
- You must set all unused bits in offset ?ISTI to 0.
- The operating system returns the file's channel number to offset ?ICH in the I/O packet. (The operating system always sets the channel number, even if you place a value in ?ICH.)
- You need not set the offsets used by ?READ and ?WRITE to 0, because the operating system ignores them when it executes ?OPEN.

Packet Versions

There are three versions of the ?OPEN parameter packet as follows. Refer to Figure 2–113 as you read the following summaries of these three versions.

Standard	which includes offsets ?ICH through ?IDEL, and is ?IOSZ words long.
Short extended	which includes offsets ?ICH through ?ENET, and is ?IBLT words long. Set bit ?IPKL in offset ?ISTI to indicate the short extended packet.
Long extended	which includes offsets ?ICH through ?ENET and 24 reserved words for a length of ?ETMX words. Set bit ?IPKL in offset ?ISTI and bit ?IMP2 in offset ?ISTO to indicate the long extended packet. You should set the 24 reserved words to zero with one exception. The first doubleword, offset ?ETER, contains zero. The second doubleword, offset ?ETSN, contains 0. The third doubleword (offset ?EPIP, the exception) contains 1S0 and a word pointer to the optional pipe extension packet. (?OPEN uses the third doubleword when working with a pipe file.)

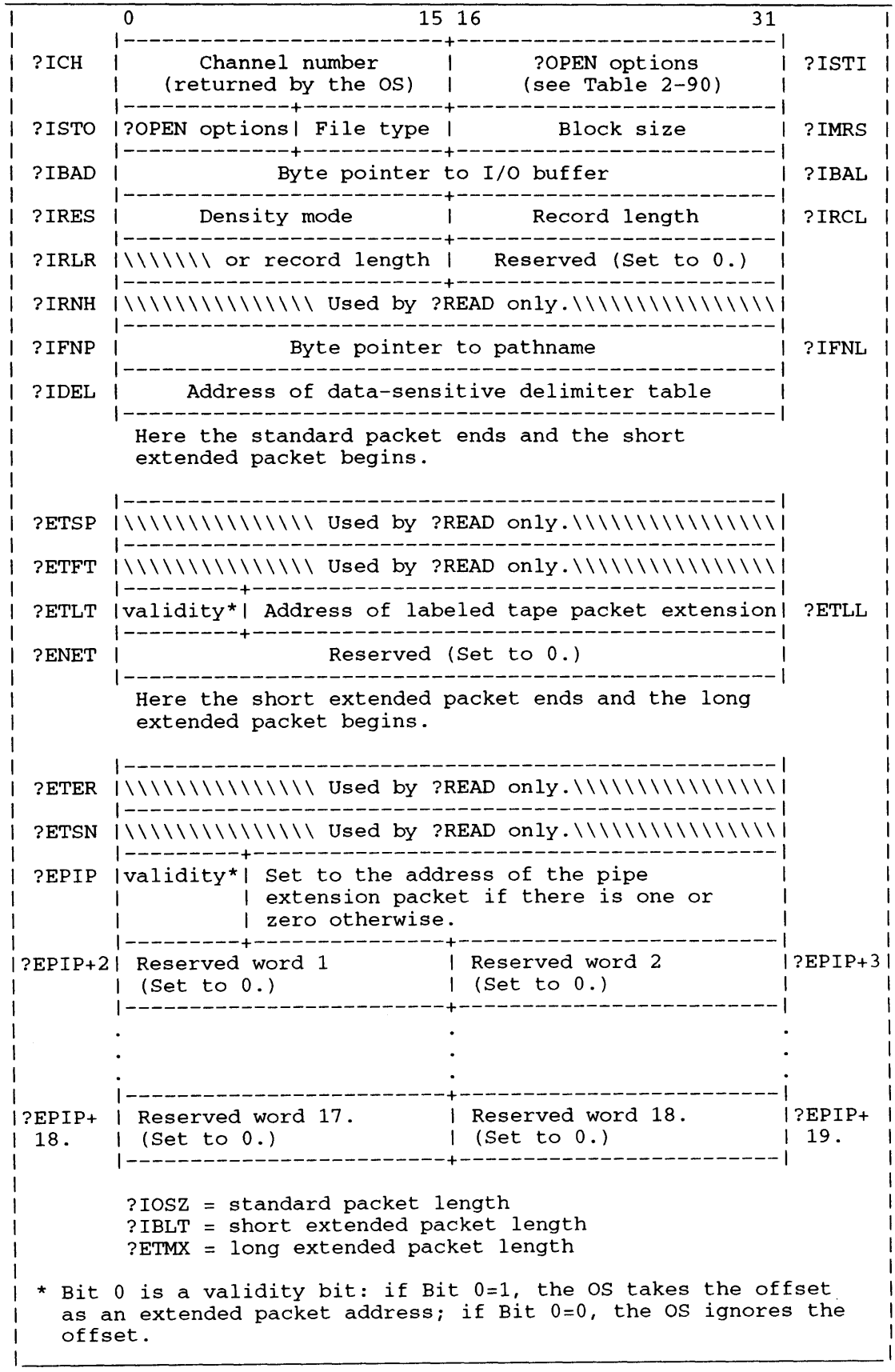


Figure 2–113. Structure of ?OPEN Packet

Table 2–90. Contents of ?OPEN Packet*

Offset	Contents
?ICH	Channel number (assigned by the OS).
?ISTI	File specifications word:
	Packet type ?IPKL--Extended packet. DEFAULT = 0 (no extended packet)
	Format select ?ICRF--Change format to that specified in record format field. DEFAULT = 0 (use record format specified in the ?CREATE packet).
	Absolute file pointer position (?READ/?WRITE only).
	Append ?APND--Open the file for appending. DEFAULT = 0 (set file pointer to first word in file).
	Binary I/O. Normally, only ?READ/?WRITE supplies a value here. However, if you issue ?OPEN against a queue type file (e.g., @LPT) and specify ?IBIN here, AOS/VS responds to this bit. Specifically, AOS/VS would do a binary queue submission to EXEC.
	Force output option (?READ/?WRITE only).
	Exclusive open ?IEXO--Permit no other task in this or any other process to open file until it has been closed. DEFAULT = 0 (nonexclusive open). Pipe files require a nonexclusive open.

* There is no default unless otherwise specified. (continued)

Table 2–90. Contents of ?OPEN Packet*

Offset	Contents
?ISTI (continued)	Priority read ?PDEL--Open file for priority read. DEFAULT = 0 (file has normal priority). Creation option ?OFCE--If the file exists, open the file. If the file does not exist, create one and open it. ?OFCR--If the file exists, return an error. If the file does not exist, create one and open it. ?OFCR!?OFCE--If the file exists, then delete it, recreate it, and open it. If the file does not exist, then create it and open it. DEFAULT = 0 (if file does not exist, do not create it; return ?OPEN error). Input/output ?OFIN--Open for input. ?OFOT--Open for output. ?OFIO--Open for input/output. (Not available for pipe files.) Record format ?RTDY--Dynamic-length. ?RTDS--Data-sensitive. ?RTFX--Fixed-length. ?RTVR--Variable-length. ?RTUN--Undefined-length. ?RTVB--Variable block, variable record. IPC no wait option ?IIPC--Setting "IPC No Wait" on IPC file When you select ?IIPC, the process that initially opens the IPC file will not wait for the synchronization message from the partner process. Normally the process waits for the partner process to open the IPC file.
?ISTO	Left byte ?SHOP--Shared ?OPEN request. (Not available for pipe files.) ?IMFF--Inhibit initial form feed. ?IMP2--Long extended packet. ?IMHN--Hold nonpriority reads. DEFAULT = 0 (normal open; write initial form feed when opening line printers). If you specify none of the options ?SHOP through ?IMHN, you must specify 0 here.

* There is no default unless otherwise specified. (continued)

Table 2–90. Contents of ?OPEN Packet*

Offset	Contents
?ISTO	Right byte File Type. (See Table 2-92) DEFAULT = 0 (if file exists, the OS ignores this parameter and uses the file type specified in the ?CREATE packet). If you are creating the file (you set ?OFCR and/or ?OFCE in ?ISTI), the file type is ?FUDF (user data file).
?IMRS	Physical block size (in Kbytes). Specify the length of the pipe in 2Kbyte (page) multiples, up to a 20 page maximum. 1 page is added to the pipe length at creation. DEFAULT = -1 (block size = 2 Kbytes)
?IBAD (doubleword)	Byte pointer to record I/O buffer. DEFAULT = -1 (deferred until you issue ?READ or ?WRITE).
?IRES	Density mode (for magnetic tapes only). Set this field to 0 for all other file and device types. ?IDAM--Automatic density matching. ?ID8--Density 800 bytes/inch. ?ID16--Density 1600 bytes/inch. ?ID62--Density 6250 bytes/inch. ?ID5--Low tape density. ?ID6--Medium tape density. ?ID7--High tape density. DEFAULT = 0 (use density mode specified during the system-generation procedure). Transfer mode (for Model 6352 magnetic tape units only) ?OMBFM--Use buffered mode when performing tape I/O. ?OMSTR--Use streaming mode when performing tape I/O. ?IMNTE--Use the emulation override. ?IMCOF--Data compression off. ?OIBM--Open as an IBM-labelled tape. ?OANS--Open as an ANSI-labelled tape. DEFAULT = 0 (open as an AOS/VS or AOS/RT32 tape).

* There is no default unless otherwise specified. (continued)

Table 2–90. Contents of ?OPEN Packet*

Offset	Contents
	9-Track tape ?INTEO -- 9-track emulation override
?IRCL	Record length. States the number of bytes to read or write for dynamic- and fixed- length records or the maximum number of bytes to read or write for data-sensitive and variable-length records. DEFAULT = -1 (defer specifying the record length until you issue a ?READ or ?WRITE. If you specify ?IRCL = -1 in the ?OPEN call, you must specify an actual record length in the ?READ or ?WRITE.
?IRLR	Number of bytes transferred (?READ/?WRITE only) or record length. See "Other ?OPEN Offsets."
?IRNW	Reserved. (Set to 0.)
?IRNH (doubleword)	Record number (?READ/?WRITE only).
?IFNP (doubleword)	Byte pointer to pathname.
?IDEL (doubleword)	Address of the delimiter table. DEFAULT = -1 (use default delimiters: null, New Line, form feed, and carriage return).
?ETLT (doubleword)	Address of the labeled magnetic tape packet extension.
?ENET (doubleword)	Reserved. (Set to 0.)
?ETER (doubleword)	Reserved. (Set to 0.)
?ETSN (doubleword)	Reserved. (Set to 0.)
?EPIP (doubleword)	Address of the pipe file extension packet. See the section "Extension Packet for Pipes."
?EPIP + 2	First of 18 reserved words. (Set to 0.)
?EPIP + 3	Second of 18 reserved words. (Set to 0.)
.	.
.	.
.	.
?EPIP + 19	Last of 18 reserved words. (Set to 0.)

* There is no default unless otherwise specified. (concluded)

Model 6352 Magnetic Tape Unit

With the exception of a Model 6351 magnetic tape unit, the following two paragraphs apply to any buffered MTJ–style tape unit including both 120 and 150 MB cartridge tape drive units and reel–to–reel tape units including Models 6586, 6587, 6588, and 6589.

In buffered mode the tape controller indicates that a tape transfer is complete after data has been read from memory, but before it has been written to tape. In this mode the system might not report error conditions for a request that fails. Therefore your program must check for errors whenever it issues a ?GCLOSE call. (The system reports all errors when it writes a file mark. Since ?GCLOSE writes a file mark, any program using buffered mode must explicitly close each tape file.)

The streaming mode allows your program to open a tape unit for high–speed backup purposes. In this mode the tape controller expects data transfer with the tape to occur quite rapidly. If data is not transferred fast enough, the performance of the tape unit degrades. So, a program that cannot maintain a high data transfer rate should not select this mode.

The File Specifications Word

As Table 2–90 indicates, you can select a number of optional file specifications in offset ?ISTI.

To select a record format different from the one specified in the ?CREATE packet, select the ?ICRF mask in offset ?ISTI and one of the record format masks. The record format masks correspond to the operating system record types: dynamic length, fixed length, data sensitive, variable length, and undefined length. For example, to define the record type as data sensitive, specify the following:

```
?ICRF!?RTDS
```

The Append Field option (?APND) moves the file pointer to the end of the file. If you select this option and then write to the file, the operating system appends the new material to the end of the file.

The exclusive open option (?IEXO) opens the file for the exclusive use of the calling task. No other task, either in the calling process or in any other process, can open the file until the current caller closes it. If you do not specify this option, you can open a file for more than one task or process to use at the same time, as long as you supply an I/O packet for each channel. (The operating system assigns a different channel number to the file on the second and subsequent ?OPENs.)

You can open a file that the operating system peripheral manager (PMGR) controls, and give its channel priority read status by selecting mask ?PDEL in offset ?ISTI. When a channel has priority read status, the operating system places that channel's ?READ requests ahead of the ?READ requests from all other channels and executes the priority reads first.

The ?OPEN packet allows you to simultaneously create and open files. Table 2–91 describes each of the file creation options for offset ?ISTI.

Table 2–91. File Creation Options for Offset ?ISTI

Mask	System Directive
?OFCE	Create the file, unless it already exists. (Ignore ?OFCE if the file already exists.)
?OFCR	Create the file; if it already exists, return an error to AC0, and pass control to the error return.
?OFCE!?OFCR	If the file already exists, delete it, and then recreate it. (You cannot use this option for IPC files.)

Table 2–92 lists the more common file types you can create with ?OPEN. Table 2–9, in the description of system call ?CREATE, lists all the file types you can create with ?OPEN.

Offset ?ISTI also contains an input/output field with three options:

- ?OFIN, which opens the file for input (?READ).
- ?OFOT, which opens the file for output (?WRITE).
- ?OFIO, which opens the file for input and/or output (?READ and/or ?WRITE).

You must select one of these options to perform I/O on the file. Otherwise, ?OPEN succeeds, but the subsequent ?READ or ?WRITE fails.

Table 2–92. Common File Types You Can Create with ?OPEN

File Type	Meaning	Comments
?FUDF	User Data File	The OS creates this type when you default the file type option (set right byte of ?ISTO to 0).
?FTXT	Text File	Should contain ASCII code.
?FPRV	32-bit Program File	An executable 32-bit program file; should contain linked, executable code.
?FDIR	Disk Directory	If you use ?OPEN to create a file of this type, you can default only the following: hashframe size, maximum number of index levels, and ACL.
?FIPC	IPC File	Directs the OS to create an IPC file or open an existing IPC file to allow full-duplex communications between two processes.
?FCPD	Control Point Directory	Although you can use ?OPEN to create a CPD, we recommend that you use ?CREATE instead.

Offset ?ISTO

The right byte of offset ?ISTO specifies the type of file you are opening. Use this field only if you are creating the file with ?OPEN. If the file you are opening already exists, the operating system ignores your input to this field and fills it with the file type you specified when you created the file.

You can set the ?SHOP parameter in offset ?ISTO to open the file for shared access. The ?SHOP parameter directs the operating system to read the file that you specify into one or more shared pages for access by more than one process. When you set the ?SHOP parameter, the operating system automatically performs an ?SOPEN on the file, and then initiates the standard ?SPAGE and ?RPAGE operations.

Note that you cannot set the ?SHOP parameter if you are creating or recreating the file with this ?OPEN. Also, note that if you open the file with a standard ?OPEN, the shared file facility does not protect your file from modifications.

The ?SHOP parameter causes ?OPEN to behave much like ?SOPEN. However, unlike ?SOPEN, ?OPEN with ?SHOP set involves the Agent. Therefore, to minimize system overhead, you should use ?SOPEN. Using ?SHOP shared I/O consumes more of the system I/O resources (than unshared I/O) and you may not be able to open as many channels.

Normally, the operating system generates a form feed when it opens a line printer. If you do not want this initial form feed, set bit ?IBFF in offset ?ISTO.

Other ?OPEN Offsets

Offset ?IBAD points to the I/O buffer. The I/O buffer is an area that you set up to receive the records you will later read or write. The I/O buffer must be large enough to accommodate the largest record you will read or write; if it is not, the operating system returns error code ERMPR when it tries to perform the I/O system call. If you do not specify an I/O buffer when you open the file, you must do so every time you issue ?READ or ?WRITE.

The meaning of offset ?IRCL varies, depending on the file's record format. If the file consists of fixed–length records, set ?IRCL to the record length in bytes. If the file consists of dynamic–length, variable–length or data–sensitive records, set ?IRCL to the number of bytes to be transferred (read or written) upon each I/O request. If the file consists of fixed–length records and ?IRCL contains –1, ?OPEN places the length of each record in the file in offset ?IRLR.

If the record type is data–sensitive and the operating system encounters a delimiter before it reaches the specified number of bytes, it terminates the I/O transfer. However, if the record type is data–sensitive and the operating system does not find a delimiter within ?IRCL bytes, it returns error code ERLTL ("line too long").

Setting a Delimiter Table

To define alternative delimiters for data–sensitive records, set up a delimiter table in your logical address space, and specify its address in the ?OPEN offset ?IDEL.

The delimiter table must consist of 16 consecutive 16–bit words, to form a table of 256 bits. Each bit in the table represents an ASCII character. Reading from left to right, the first bit (Word 0, Bit 0) represents the null character (0), the second bit (Word 0, Bit 1) represents the Ctrl–A character (octal 001), and so forth. For each bit you set, the operating system recognizes the corresponding character as a delimiter for the file's records.

Figure 2–114 depicts a sample delimiter table with bits set to make the null, carriage return (015 octal), and rubout (177 octal) characters data–sensitive delimiters.

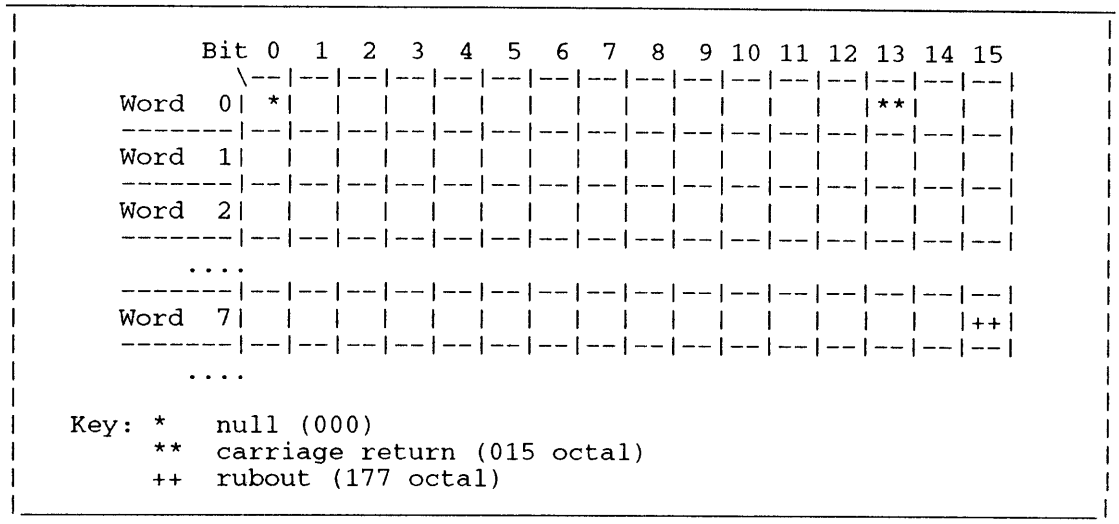


Figure 2–114. Sample Delimiter Table

You can also use the ?SDLM system call to set a delimiter table for an open record or open device. If you issue ?SDLM after you issue ?OPEN, the ?SDLM delimiter specifications override those in the ?OPEN packet.

Extension Packet for Pipes

You use one extension packet for pipes, and only ?OPEN provides the pipe extension packet. You supply the packet as an extension to the packet for a ?OPEN, ?READ, or ?WRITE system call. You must specify the pipe size in 2 Kbyte multiples in offset ?IMRS (see Table 2–90). The maximum pipe size is 20 pages (2 Kbytes equal one page), or you can use the default value of –1.

When the pipe file is created, the operating system adds a 1–page overhead to the size specified in ?IMRS. For a full description of using pipes, see *AOS/VS System Concepts* (093–000335).

Figure 2–115 contains the extension packet. Table 2–93 describes its contents.

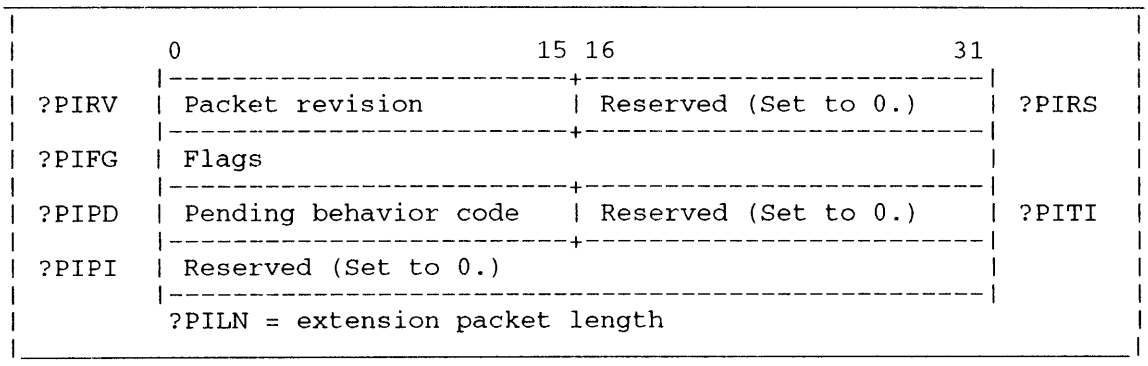


Figure 2–115. Structure of ?OPEN Extension Packet for Pipes

Table 2–93. Contents of ?OPEN Extension Packet for Pipes

Offset	Contents
?PIRV	Packet revision number. Place the value of symbol ?PKR0 here.
?PIRS	Reserved word. (Set to 0.)
?PIFG	Flag word. (Set to 0.)
?PIPD	Use the following symbols to specify the operating system's action. ?PALW -- Always pend on a read from an empty pipe or on a write to a full pipe, regardless of whether the pipe is one-ended or two-ended. ?PTWO -- Pend on a read from an empty pipe or on a write to a full pipe, but only if the pipe is two-ended. If the pipe is one-ended, then return error code EREOF on a read or else ERPFL on a write. ?PTWO is the default value of this offset. ?PNVR -- Never pend on a read from an empty pipe or on a write to a full pipe, regardless of whether the pipe is one-ended or two-ended. Return error code EREOF on a read or else ERPFL on a write.
?PITI	Reserved word. (Set to 0.)
?PIPI (double-word)	Reserved word. (Set to 0.)

Magnetic Tapes

Offset ?IMRS specifies the physical block size of a magnetic tape or disk file. The physical block size is the number of bytes, minus 1, in the largest physical block. The operating system uses this parameter to allocate a buffer ?IMPSTL bytes long for I/O. (The default buffer size is 2 Kbytes (4 blocks) for disk files.)

If you are opening a magnetic tape on a controller that supports multiple tape densities (such as a type MTB controller), you can set offset ?IRES to a specific density mode for the tape. There are seven density mode settings:

- ?IDAM directs the operating system to set the tape unit to the correct density mode automatically.
- ?ID8 sets the tape unit to a density mode of 800 bytes/inch.
- ?ID16 sets the tape unit to a density mode of 1600 bytes/inch.
- ?ID62 sets the tape unit to a density mode of 6250 bytes/inch.
- ?ID5 sets the tape unit to the lowest density it supports.
- ?ID6 sets the tape unit to the middle density on drives supporting three densities, and to the lower density on drives supporting two densities.
- ?ID7 sets the tape unit to the highest density it supports.

If you set offset ?IRES to 0 (the default), the operating system uses the density mode that was set during the system–generation procedure. However, if you set offset ?IRES to 0, you must make sure that the tape's density mode matches the default mode. Otherwise, ?OPEN fails after I/O is attempted.

There are two other possible density mode errors: ERCND (controller does not support this density mode), and ERITD (indecipherable tape density). The operating system returns ERITD when either the tape or the tape unit is damaged.

Labeled Magnetic Tapes

If you want to read or write additional user labels on a labeled tape file, you must open the file with the ?OPEN packet extension for labeled tapes. To do this, perform the following steps:

1. Set Bit ?IPKL in offset ?ISTI of the standard ?OPEN packet.

2. Set Bit 0 of offset ?ETLT to 1 (to indicate that there is a packet extension).

   The first time you use the labeled tape extension, you must set Bit 0 of ?ETLT (the flag bit) to 1 so that the system will recognize Bits 1 through 31 of offset ?ETLT as the address of the packet extension. The operating system sets the flag bit to 0 after it executes the ?OPEN.

   If you want to change one or more of the specifications, and then reuse the extension, be sure to reset the flag bit to 1. If the flag bit is 0, the operating system assumes that the packet extension has not changed and, therefore, does not re–examine it.

   You can set aside space for the labeled tape extension even if you do not supply values for it. In this case, the operating system simply fills the packet extension with the existing label information.

3. Set Bits 1 through 31 of offset ?ETLT to the address of the packet extension.

4. Precede the labeled tape extension with the following pseudo–op to reserve the correct number of words for the packet extension:

   ```
   .LOC  extended–pkt–address+?ELLN
   ```

Figure 2–116 shows the structure of the labeled tape extension.

If you create and open a labeled tape file with the ?OPEN creation option, the operating system treats the extension as follows:

- If you supplied values in the extension, the operating system uses those values.
- If you did not supply values in the extension, the operating system fills the packet extension with the default values listed in Table 2–94.

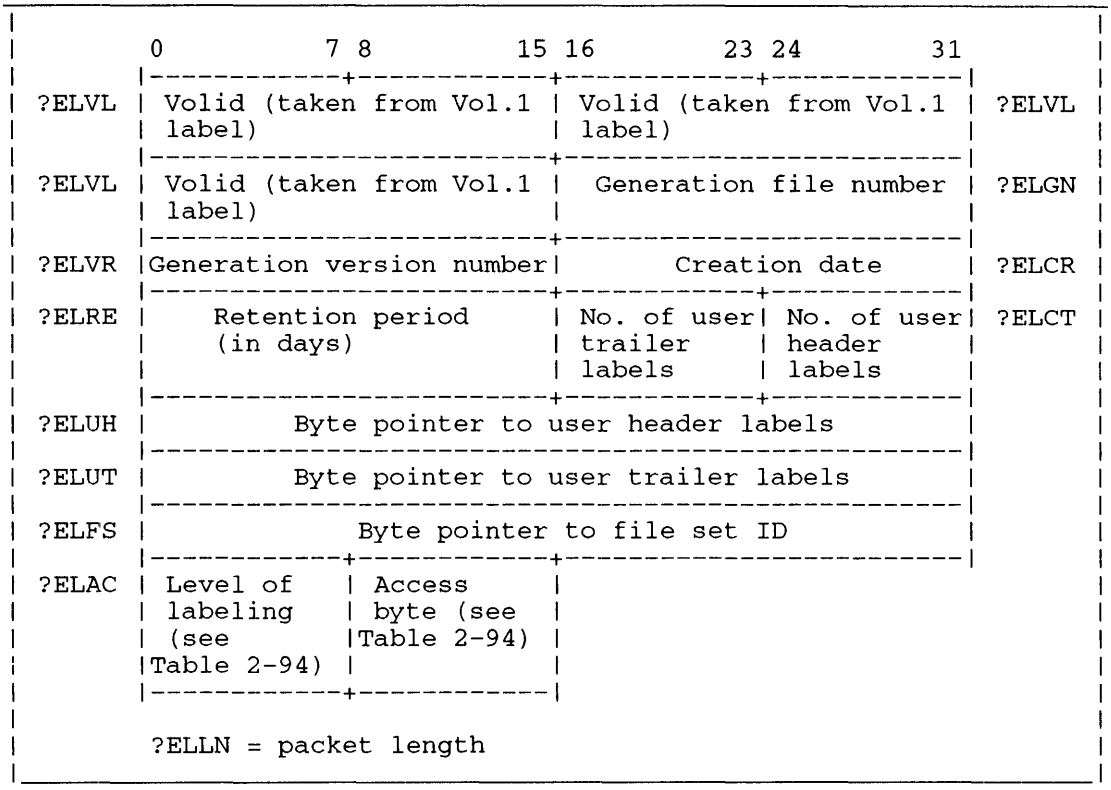


Figure 2–116. Structure of Labeled Magnetic Tape Packet Extension

Table 2–94. Contents of Labeled Magnetic Tape Packet Extension*

Offset	Contents
?ELVL	Volid (returned by the OS) taken from Volume 1 Label 3 words.
?ELGN	Generation number of file (from 0 through 9999). DEFAULT = -1 (if you specify -1, the OS uses 0001 when it writes to the file, and does not check or return the generation number when it reads the file).
?ELVR	Generation version number (from 0 through 99). DEFAULT = -1 (if you specify -1, the OS uses 00 when it writes to the file, and does not check or return the version number when it reads the file).
?ELCR	Creation date in standard form: dd.mm.yy. DEFAULT = current system date.
?ELRE	Retention period in days. DEFAULT = 90 days.
?ELCT	Right byte: number of user header labels, within the range 1 through 9 (0 means none). Left byte: number of user trailer labels, within the range 1 through 9 (0 means none). At ?OPEN time, the OS either reads or writes the header labels, depending on whether you opened the file for input (reading) or for output (writing). When you close the labeled tape file, the OS writes the user trailer labels from the area specified in offset ?ELUT. If the file was not modified, the OS reads the trailer labels into the area specified in ?ELUT.

* There is no default unless otherwise specified. (continued)

Table 2–94. Contents of Labeled Magnetic Tape Packet Extension*

Offset	Contents
?ELUH (doubleword)	Byte pointer to an area containing the header labels; all labels are contiguous and sequential.
?ELUT (doubleword)	Byte pointer to an area containing the trailer labels; all labels are contiguous and sequential.
?ELFS (doubleword)	Byte pointer to file set ID; if you set ?ELFS to -1, the OS ignores the file set ID when it reads, and uses the default when it writes. DEFAULT = 0 (Use volid for first volume in file set.)
?ELAC	Level of labeling (left byte) ?ELL1--Level 1 (ANSI or IBM). ?ELL2--Level 2 (ANSI or IBM). ?ELL3--Level 3 (ANSI). DEFAULT = 0 (if ANSI format, ANSI Level 3; if IBM format, IBM Level 2). Access byte (right byte): For ANSI format, set this byte to one or more characters that specify access to the tape. (For IBM format, set this byte to 0.) DEFAULT = 0 (if ANSI format, blank space; if IBM format, 0).

* There is no default unless otherwise specified. (concluded)

Note that there is no default value for the tape volume identifier (volid). If you want to create and open a labeled tape file and default the labeled tape extension, you must first establish a volid for the tape volume. You can do this with the ?LABEL system call or with the CLI LABEL utility.

You can create files on any labeled tape volume as long as you specify the correct volid. If the file does not exist, the operating system creates it after the last file.

If the file exists, but its retention period has expired and you selected the delete and create options in the ?OPEN packet, the operating system overwrites the existing file with the new file. Note that the delete and create option also deletes all subsequent files on the tape, because magnetic tape files are sequential.

When you open a labeled magnetic tape, do not use the ?OPEN option ?OFIO in offset ?ISTI, which opens the file for both reading and writing. If you use this option, the ?OPEN fails and the operating system returns error code ERIOO (illegal option for open type) in AC0.

You can use as many as nine user header labels and nine user trailer labels. Each user header and trailer label is 80 bytes (characters) long, where the last 76 bytes are user data. If the operating system finds a null character when writing the label, it takes the null as a terminator, and if necessary, pads the label with spaces to fill out the 80 character positions. The next label starts 80 bytes past the first label. When the operating system reads the label, it returns the data from the first label to the first 76 bytes of the ?ELUT area, the second label to the second 76 bytes, and so forth.

If you intend to write to the labeled tape file, be sure to set the following fields in the file's ?OPEN packet:

- Record format in offset ?ISTI (with the flag ?ICRF; for example, ?RTFX!?ICRF for fixed–length records).
- Record length in offset ?IRCL.
- Block length in offset ?IMRS.

These fields must contain valid information; otherwise, another operating system may not be able to read the tape.

When the operating system writes to a labeled tape file, it will overwrite the existing file data and all subsequent data on the tape, if the retention period (?ELRE) has expired. The default retention period is 90 days.

If you want to read a file on the labeled tape, do not set ?ICRF when you set the record format, because the operating system will be unable to return the record format to ?ISTI after it reads the file.

The operating system returns the file's record length to ?IRCL after it reads a labeled magnetic tape file, and returns the block length to ?IMRS. If you want the operating system to return the block length after a ?READ, be sure to set ?IMRS to –1 in the ?OPEN packet.

Sample Packet

The following sample packet opens a file with data–sensitive records for file I/O. The file already exists.

```
PKT:    .BLK     ?IBLT           ;Allocate enough space for packet.
                                 ;Packet length = ?IBLT.
        .LOC     PKT+?ICH        ;Channel number.
        .WORD    0               ;The OS returns the channel number.
        .LOC     PKT+?ISTI       ;?OPEN options.
        ?ICRF!?RTDS!?OFIO        ;Change record format to that
                                 ;specified in record format field
                                 ;(?ICRF), which is data-sensitive
                                 ;(?RTDS), and open for I/O (?OFIO).
        .LOC     PKT+?ISTO       ;File type.  The OS assumes the file
        .WORD    0               ;type that you specified when you
                                 ;created the file.
        .LOC     PKT+?IMRS       ;Physical block size is
        .WORD    -1              ;2 Kbytes (-1 is the default).
        .LOC     PKT+?IBAD       ;Specify byte pointer to record I/O
                                 ;buffer.
        .DWORD   BUF*2           ;Byte pointer to BUF.
        .LOC     PKT+?IRES       ;Reserved.
        .WORD    0               ;You must set this value to 0.
        .LOC     PKT+?IRCL       ;Specify record length.
        .WORD    120.            ;Maximum record length is 120.
                                 ;characters.
        .LOC     PKT+?IFNP       ;Specify byte pointer to pathname.
        .DWORD   PTH*2           ;Byte pointer to PTH.
        .LOC     PKT+?IDEL       ;Specify delimiter table address.
        .DWORD   -1              ;Use default delimiters: New Line,
                                 ;form feed, and carriage return (-1
                                 ;is the default).
        .LOC     PKT+?IBLT       ;End of packet.
```

The following sample packet deletes a file with fixed-length records, recreates it, and then opens it for output.

```
PKT:    .BLK    ?IBLT           ;Allocate enough space for packet.
                                ;Packet length = ?IBLT.
        .LOC    PKT+?ICH        ;Channel number.
        .WORD   0               ;The OS returns the channel number.
        .LOC    PKT+?ISTI
        .WORD   ?ICRF!?RTFX!?OFCR!?OFCE!?OFOT   ;Change record format
                                ;to that specified in record format
                                ;field (?ICRF), which is fixed-length
                                ;(?RTFX), delete, and then recreate
                                ;the file (?OFCR!?OFCE), and open the
                                ;file for output (?OFOT).
        .LOC    PKT+?ISTO       ;Specify file type.  The OS assumes
        .WORD   0               ;that it is a user data file (?FUDF)
                                ;by default.
        .LOC    PKT+?IMRS       ;Physical block size is
        .WORD   -1              ;2 Kbytes (-1 is the default).
        .LOC    PKT+?IBAD       ;Specify byte pointer to record I/O
                                ;buffer.
        .DWORD  BUF*2           ;Byte pointer to BUF.
        .LOC    PKT+?IRES        ;Reserved.
        .WORD   0                ;You must set this value to 0.
        .LOC    PKT+?IRCL        ;Specify record length. (?READ and
                                 ;?WRITE use this value.)
        .WORD   33.              ;Maximum record length is 33-byte
                                 ;fixed records.
        .LOC    PKT+?IRLR        ;Record length.  (?READ and ?WRITE
                                 ;use this value.)
        .WORD   0                ;The OS returns this value.
        .LOC    PKT+?IFNP        ;Specify byte pointer to pathname.
        .DWORD  PTH*2            ;Byte pointer to PTH.
        .LOC    PKT+?IDEL        ;Specify delimiter table address.
        .DWORD  -1               ;Use default delimiters: New Line,
                                 ;form feed, and carriage return
                                 ;(-1 is the default).
        .LOC    PKT+?IBLT        ;End of packet.
```

Notes

- See the descriptions of ?CLOSE, ?CREATE, ?READ, ?WRITE, ?SPAGE, ?RPAGE, ?SDLM, and ?LABEL in this chapter.
- See the description of ?CGNAM in this chapter to get a complete pathname from a channel number.

AOS/VS

?OPER *[packet address]*
error return
normal return

Input

AC0	Reserved (Set to 0.)
AC1	Reserved (Set to 0.)
AC2	Address of the ?OPER packet, unless you specify the address as an argument to ?OPER

Output

AC0	Unchanged
AC1	Unchanged
AC2	Address of the ?OPER packet

Error Codes in AC0

ERBTL	Buffer too long
ERDAE	Operator already exists
ERDEB	Daemon resource error in error buffer
ERDNE	Task is not an operator
ERDRF	Daemon resource failure
ERIRB	Insufficient room in buffer
ERNOA	No operator available
ERNRQ	No matching operator request
ERPKT	Invalid packet identifier
ERPNO	Process is not an operator
ERPRE	Invalid system call parameter
ERVWP	Invalid word pointer passed as system call argument

Why Use It?

Use this system call to communicate with an operator and receive its responses.

Who Can Use It?

There are no special process privileges needed to issue this call, and there are no restrictions concerning file access.

What It Does

This system call performs six basic functions. They are

- Operator on.
- Operator off.
- Send operator request.
- Operator receive request.
- Operator respond to request.
- Operator information.

You supply a main packet in all operator interface requests. You also supply a subpacket that the main packet points to. This subpacket contains the details about one of the six functions mentioned above.

Figure 2–117 shows the structure of the ?OPER main packet, and Table 2–95 describes its contents. After Table 2–95 the subpackets are described in the following order.

Function Code	Function Description	Structure of Subpacket	Contents of Subpacket
?OPON	Operator on	Figure 2–118	Table 2–96
?OPOFF	Operator off	Figure 2–119	Table 2–97
?OPSEND	Send operator request	Figure 2–120	Table 2–98
?OPRCV	Operator receive request	Figure 2–121	Table 2–99
?OPRESP	Operator respond to request	Figure 2–122	Table 2–100
?OPINFO	Operator information	Figure 2–123	Table 2–101

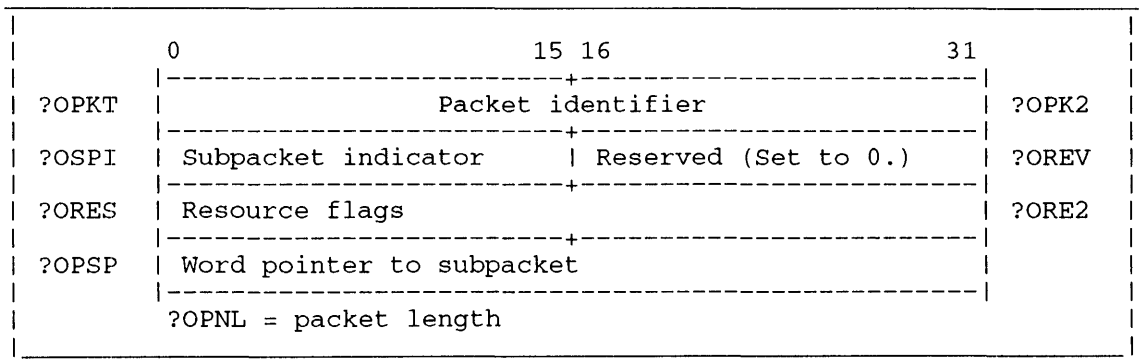


Figure 2–117. Structure of ?OPER Main Packet

Table 2–95. Contents of ?OPER Main Packet

Offset	Contents
?OPKT/ ?OPK2	Packet identifier. Place ?OPID here.
?OSPI	Function code. It specifies the operation you want, and indicates the existence of a unique subpacket to the operating system. You must supply this main packet and exactly one subpacket for each issuance of ?OPER. Select from the following values: ?OPON -- Become an operator daemon. ?OPOFF -- Resign as an operator daemon. ?OPINFO -- Request information about the current operator environment. ?OPSEND -- Issue an operator request. ?OPRCV -- Receive an operator request. ?OPRESP -- Respond to an operator request.
?OREV	Reserved; set to 0.
?ORES/ ?ORE2	Resource indicator bit flags. This doubleword contains device bit indicators that represent the various system resources. Each indicator corresponds to the resource(s) the function is intended for. The bit masks and their resources are as follows: ?OPLD -- Labeled diskettes (LD). ?OPUD -- Unlabeled diskettes (UD). ?OPLT -- Labeled tapes (LT). ?OPUT -- Unlabeled tapes (UT). ?OPCO -- Terminals (T). ?OPPR -- Printers (P). ?OPBQ -- Batch queues (B). ?OPHD -- Hard disks (HD). ?OPQU -- Queues (Q). ?OPCD -- Other cooperative devices (OCD). ?OPGM -- Generic media (GM). ?OPMI -- Miscellaneous (M). For example, to specify terminals and printers, supply the symbol ?OPCO!?OPPR.
?OPSP	Word pointer to subpacket. Supply a word pointer to the subpacket for the function you selected in offset ?OPKT/?OPK2.

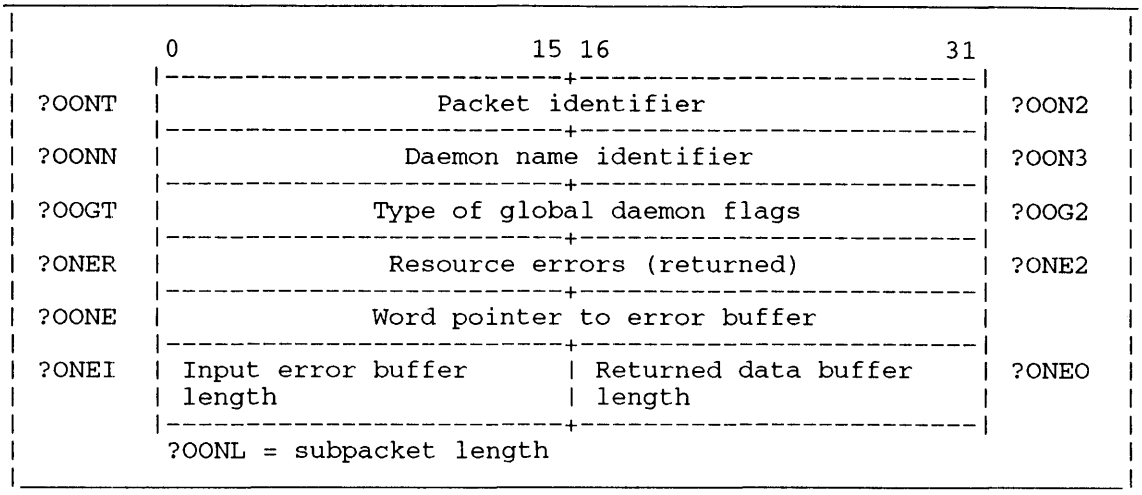


Figure 2–118. Structure of ?OPON Subpacket

Table 2–96. Contents of ?OPON Subpacket

Offset	Contents
?OONT/ ?OON2	Packet identifier. Place ?ONID here.
?OONN/ ?OON3	Daemon name. This offset contains the name that uniquely identifies the global daemon(s) being established for the resources that offset ?ORES in the main packet indicates. ?OONN/?OON3 is 32 bits long; you must supply a value here when you declare any type of a global daemon. Set this field to zero when you are making a request for declaring a local daemon.
?OOGT/ ?OOG2	Flags word. This doubleword identifies the type of global daemon that you want to become for each resource that offset ?ORES of the main packet specifies. When you want to become an exclusive operator daemon for a resource, set the bit indicator flag corresponding to the resource bit flag in offset ?ORES. Doing so declares exclusive operator global-type daemons for specified resources simultaneously with requests to declare global operator-type daemons for other resources. If you want to become a local daemon, then set this field to zero.
?ONER/ ?ONE2	Resource error. The operating system returns the success or failure of your becoming a daemon for the specified resources in offset ?ORES/?ORE2 of the main packet. If you are unsuccessful, then the operating system sets the device bit indicator corresponding to the failed resource in ?ORES/?ORE2. If you are successful for all resources requested, then the operating system sets offset ?ONER/?ONE2 to zero.

(continued)

Table 2–96. Contents of ?OPON Subpacket

Offset	Contents
?OONE (double-word)	Error buffer pointer. Supply the word address of the error data buffer. The operating system returns error codes to this buffer that indicate why it did not declare the daemon for the indicated resources in offset ?ONER/?ONE2 above. Each system resource has a dedicated word entry in the buffer to hold the encountered error codes. If the operating system detects an error while declaring a daemon for a particular resource, it loads the appropriate error code into the correct entry. If you don't want particular errors, then set this field to zero before issuing ?OPER. Here is the resource error buffer word entry format: 1. error code for labeled diskettes 2. error code for unlabeled diskettes 3. error code for labeled tapes 4. error code for unlabeled tapes 5. error code for terminals 6. error code for printers 7. error code for batch queues 8. error code for hard disks 9. error code for queues 10. error code for other cooperative devices 11. error code for generic media 12. error code for miscellaneous
?ONEI	Error buffer length. Supply the word length of the error buffer. The buffer must be large enough to accommodate all of the defined system resources, so place at least 12 here. If offset ?OONE contains zero, the system ignores this word.
?ONEO	Returned error buffer length. The system contains the word length of the data buffer returned. The system ignores this offset if offset ?OONE contains zero.

(concluded)

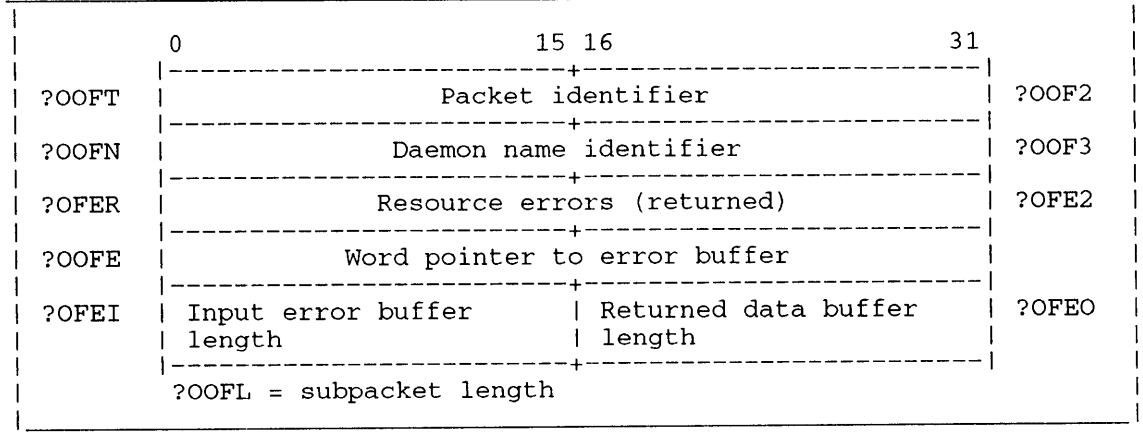


Figure 2–119. Structure of ?OPOFF Subpacket

Table 2–97. Contents of ?OPOFF Subpacket

Offset	Contents
?OOFT/ ?OOF2	Packet identifier. Place ?OFID here.
?OOFN/ ?OOF3	Daemon name. This offset contains the same name identifier for each global daemon that is requesting to resign. The identifier is limited to 32 bits; it is the same identifier that was set in the ?OPON function. Set this field to zero when the daemon that wants to resign is a local daemon.
?OFER/ ?OFE2	Resource error. The operating system returns the success or failure of resigning a daemon for the specified resources in offset ?ORES/?ORE2 of the main packet. If you are unsuccessful, then the operating system sets the device bit indicator corresponding to the failed resource in ?ORES/?ORE2. If you are successful for all resources requested, then the operating system sets offset ?OFER/?OFE2 to zero.
?OOFE (double-word)	Error buffer pointer. Supply the word address of the error data buffer. The operating system returns error codes to this buffer that indicate why it did not resign the daemon for the indicated resources in offset ?OFER/?OFE2 above. Each system resource has a dedicated word entry in the buffer to hold the encountered error codes. If the operating system detects an error while resigning a daemon for a particular resource, it loads the appropriate error code into the correct entry. If you don't want particular errors codes, then set this field to zero before issuing ?OPER.

(continued)

Table 2–97. Contents of ?OPOFF Subpacket

Offset	Contents
	Here is the resource error buffer word entry format: 1. error code for labeled diskettes 2. error code for unlabeled diskettes 3. error code for labeled tapes 4. error code for unlabeled tapes 5. error code for terminals 6. error code for printers 7. error code for batch queues 8. error code for hard disks 9. error code for queues 10. error code for other cooperative devices 11. error code for generic media 12. error code for miscellaneous
?OFEI	Error buffer length. Supply the word length of the error buffer. The buffer must be large enough to accommodate all of the defined system resources, so place at least 12 here. If offset ?OOFE contains zero, the system ignores this word.
?OFEO	Returned error buffer length. The system contains the word length of the data buffer returned. The system ignores this offset of offset ?OOFE contains zero.

(concluded)

Table 2–98. Contents of ?OPSEND Subpacket

Offset	Contents
?OSNT/ ?OSN2	Packet identifier. Place ?SEID here.
?OSNP (double-word)	Data buffer pointer. This offset contains the word address of the data buffer. The buffer transfers data that is associated with particular operator requests. The operator daemon specifies the format of the data.
?OSIL	Supply the word length of the data buffer.
?OSOL	The operating system returns the word length of data returned to you in the buffer.
?OSID	Supply the number of words of input data in the buffer.
?OSNQ	Supply the request code for the function you are sending to the local daemon. Here is a list of the codes with their resources and functional descriptions: ?ORLC (LD) -- labeled diskette close. ?ORLO (LD) -- labeled diskette open. ?ORMNV (LD) -- mount next volume. ?ORMU (LD) -- mount error.
?OSNN/ ?OSN3	Daemon identifier. For sending requests to local daemons, set this offset to zero and place ?OSNL in offset ?OSNF. For sending requests to a specific global daemon, supply the name identifier of the global daemon and place ?OSNG in offset ?OSNF. For sending requests to any available global daemon, set this offset to zero and place ?OSNO in offset ?OSNF. The operating system returns the name identifier of a global daemon. You could then use this name identifier in subsequent ?OPSEND requests.
?OSNF	Flags word. This offset contains flag bits with routing information for the ?OPSEND function. Set the flag bit that corresponds to the routing schemes as follows: ?OSNL -- send to the local daemon for the resource in offset ?OSNQ. ?OSNG -- send to a specific global daemon, whose name is in offset ?OSNN/?OSN3, for the resource in offset ?OSRE of the main packet. ?OSNO -- send to any available global daemon for the resource; the operating system returns, in offset ?OSNN/?OSN3, the name identifier of the global daemon where it routed the resource.
?OSNR	Reserved; set to 0.

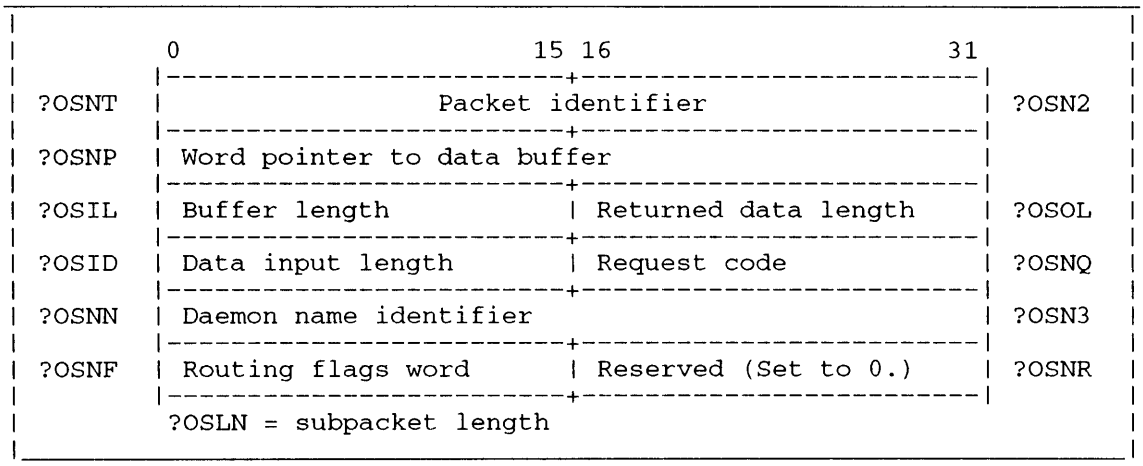


Figure 2–120. Structure of ?OPSEND Subpacket

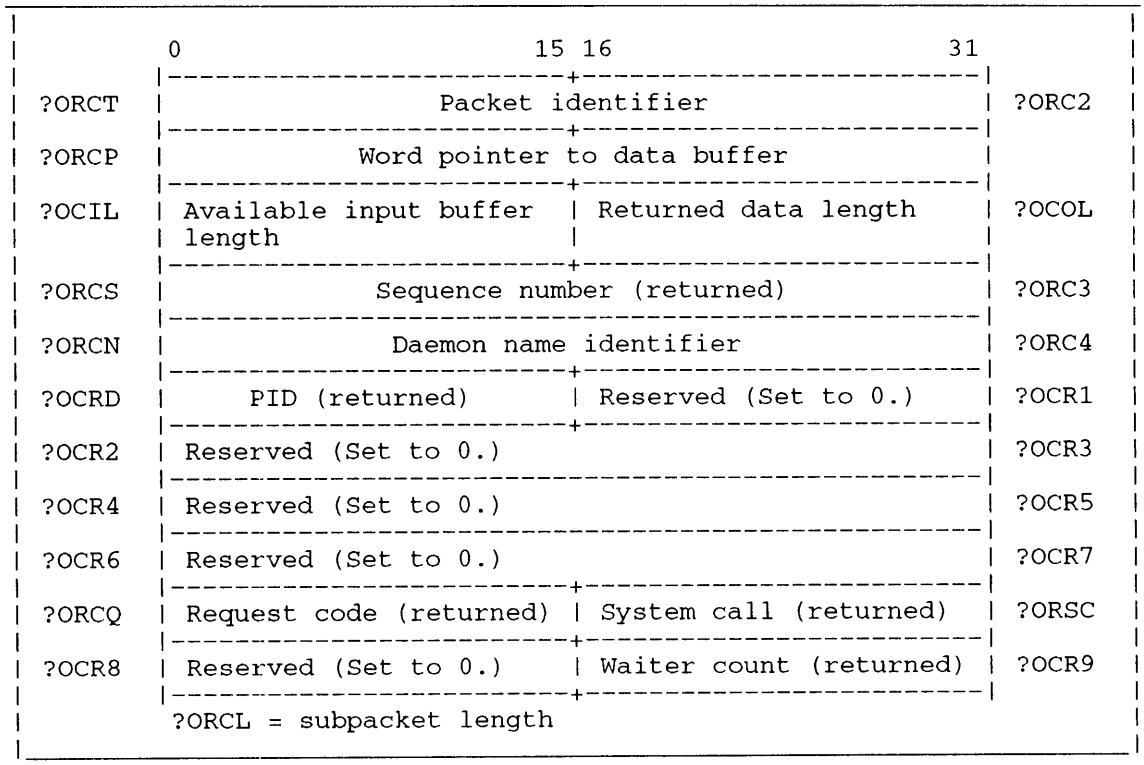


Figure 2–121. Structure of ?OPRCV Subpacket

Table 2–99. Contents of ?OPRCV Subpacket

Offset	Contents
?ORCT/ ?ORC2	Packet identifier. Place ?RCID here.
?ORCP (double-word)	Data buffer pointer. This offset contains the word address of the data buffer. The buffer transfers data that is associated with particular operator requests. The operator daemon specifies the format of the data.
?OCIL	Input buffer length. This offset contains the word length of the input data buffer.
?OCOL	Output buffer length. This offset contains the word length of data returned to you in the buffer.
?ORCS/ ?ORC3	Sequence number. The operating system returns a unique sequence identifying number. The number is used to match responses with requests from callers.
?ORCN/ ?ORC4	Daemon name. Supply the name identifier of the calling daemon. If it is a global daemon, then supply the associated name identifier. If it is a local daemon, then supply zero.
?OCRD	PID. The operating system returns the PID of the sender process.
?OCR1	Reserved. (Set to 0.)
?OCR2/ ?OCR3	Reserved. (Set to 0.)
?OCR4/ ?OCR5	Reserved. (Set to 0.)
?OCR6/ ?OCR7	Reserved. (Set to 0.)
?ORCQ	The operating system returns the request code for the function that was sent to the daemon. The codes and their meanings are the same as those in offset ?OSNQ of the ?OPSEND subpacket.
?ORSC	System call. AOS/VS returns the system call number that was issued by the requestor (?EXEC, ?OPEX, or ?OPER).
?OCR8	Reserved. (Set to 0.)
?OCR9	Waiter count. AOS/VS returns the number of outstanding requests that are queued for this daemon. If the count is zero, the daemon had to wait for a request. Otherwise, the count contains the number of outstanding send requests that were not received at the time of the call. Use this offset to return performance information.

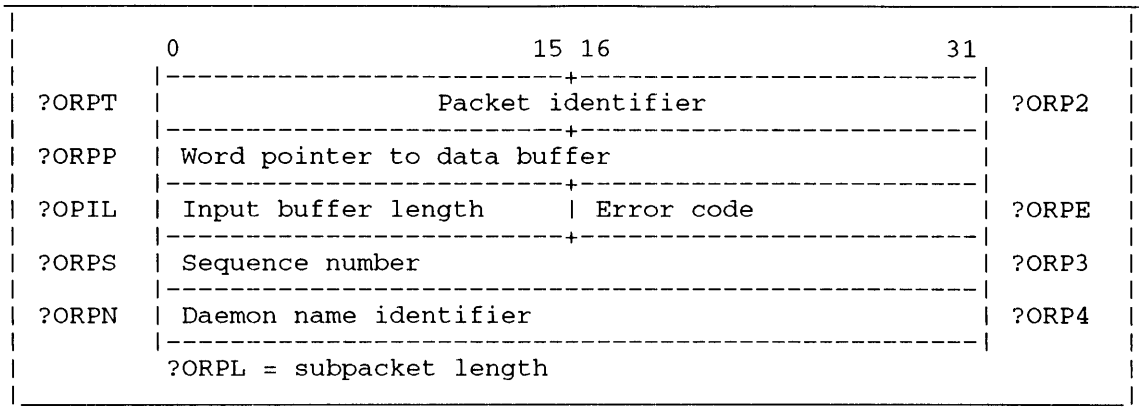


Figure 2–122. Structure of ?OPRESP Subpacket

Table 2–100. Contents of ?OPRESP Subpacket

Offset	Contents
?ORPT/ ?ORP2	Packet identifier. Place ?RSID here.
?ORPP (double-word)	Data buffer pointer. This offset contains the word address of the data buffer. The buffer transfers data that is associated with particular operator requests. The daemon specifies the format of the data. Moreover, the buffer holds data that the daemon is returning to a process; it's the process that issued ?OPER/?OPSEND for this particular request.
?OPIL	Input buffer length. Supply the number of words in the input data buffer.
?ORPE	Error code. The calling daemon uses this field to notify the operating system of any errors detected during the request's processing. If no errors were encountered, then set this field to zero. Otherwise, set it to the code for the detected error.
?ORPS/ ?ORP3	Sequence number. This offset contains the unique sequence ID number that the daemon is responding to. This must be the same sequence number that the ?OPRCV function returned. This offset is required on input from the daemon, and the operating system uses this offset to map responses to the correct requestor.
?ORPN/ ?ORP4	Daemon name. Supply the name identifier of the calling daemon. If this daemon is a global type, supply the associated name identifier. If this daemon is a local one, place zero here.

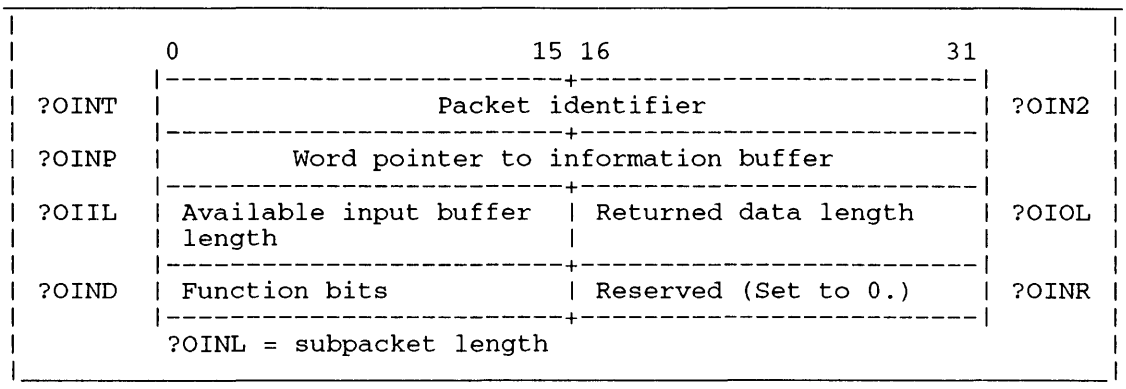


Figure 2–123. Structure of ?OPINFO Subpacket

Table 2–101. Contents of ?OPINFO Subpacket

Offset	Contents
?OINT/ ?OIN2	Packet identifier. Place ?INID here.
?OINP (double-word)	Information buffer pointer. Supply the word address of a buffer that receives information from the ?OPINFO operator function. Information returned to you includes: - A bit map indicating which resources have global daemons. - A bit map indicating which of the resources have exclusively declared global daemons. - The name identifiers of daemons for a specific resource.
?OIIL	Input buffer length. This offset contains the number of words in the buffer addressed by offset ?OINP. Supply a large enough number so the buffer can hold the identifiers for all system resources.
?OIOL	Output buffer length. The operating system returns the number of words of data that it returned in the buffer.
?OIND	Function bit indicators. Supply bits to indicate the type of information you want returned to the buffer. The values of the bits and their meanings are ?OIGB--return the bit maps of system resources having a global declared daemon, and indicate which of these daemons are exclusively declared global. The operating system returns the bit maps in the first two doublewords of the buffer. The first doubleword indicates which resources have global daemons declared. The second doubleword indicates which of the existing global daemons are exclusively declared. The system sets corresponding bit flags. These bits are the same as in the ?OPON packet(s). The first doubleword will contain resource bits; the second will contain exclusive resource bits. ?OIGN--return the identifiers of global daemons that exist for the resource whose bit mask is set in offset ?ORES/ORE2 of the main packet. The identifiers are returned in the buffer that offset ?OINP points to.
?OINR	Reserved. (Set to 0.)

?OPEX — Communicates between the current process and an operator process.

AOS/VS

?OPEX *[packet address]*
error return
normal return

Input

AC0	Reserved (Set to 0.)
AC1	Reserved (Set to 0.)
AC2	Address of the ?OPEX packet, unless you specify the address as an argument to ?OPEX

Output

AC0	Unchanged
AC1	Unchanged
AC2	Address of the ?OPEX packet

Error Codes in AC0

ERBTL	Buffer too long
ERDAE	Operator already exists
ERDEB	Daemon resource error in error buffer
ERDNE	Task is not an operator
ERDRF	Daemon resource failure
ERIRB	Insufficient room in buffer
ERNOA	No operator available
ERNRQ	No matching operator request
ERPKT	Invalid packet identifier
ERPNO	Process is not an operator
ERPRE	Invalid system call parameter
ERPRV	Caller not privileged for this action
ERRVN	Reserved value not zero
ERVBP	Invalid byte pointer passed as system call argument
ERVWP	Invalid word pointer passed as system call argument
ERXSP	Invalid EXEC string parameter
ERXUF	Unknown request code

Why Use It?

Use this system call to make requests of operator daemons for actions on various system resources. The actions can be service requests or status information requests.

Who Can Use It?

To issue ?OPEX you must have one of the following:

- The same username as EXEC (which is usually OP).
- System Manager privilege.
- The appropriate ACL to the related device/queue/cooperative.

There are no restrictions concerning file access.

?OPEX Continued

What It Does

This system call performs the functions of the CONTROL @EXEC family of commands. The call's requests end up as part of an ?OPER system call parameter packet. In turn, the packet is sent to the appropriate ?EXEC daemon for the corresponding resource request type.

You supply a main packet to any ?OPEX request. This main packet includes, in offset ?ZXFU, a function code to specify the exact request. The structure of the main packet is in Figure 2–124. Table 2–102 shows the function codes that you can place in offset ?ZXFU of the main packet. This table also explains the contents of the other offsets of the main packet.

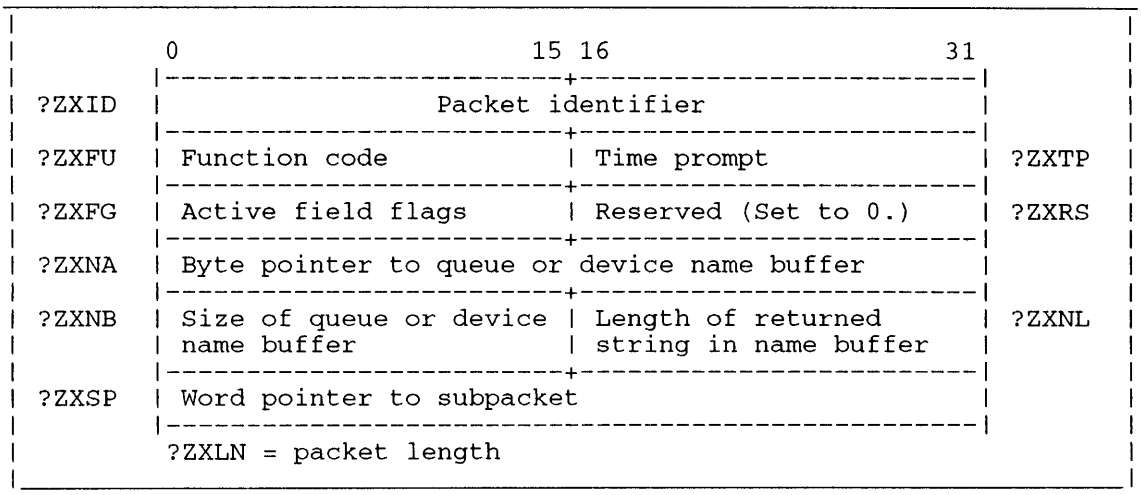


Figure 2–124. Structure of ?OPEX Main Packet

Table 2–102. Contents of ?OPEX Main Packet

Offset	Contents
?ZXID (double-word)	Packet identifier. Place ?ZXIZ here.
?ZXFU	Function code. It specifies the ?OPEX function you want. Place one of the following values in this offset.

Value	Functional Description	See Figure	See Table	See Page
?ZAC	access command	2-125	2-103	2-442
?ZAG	align command	2-126	2-104	2-443
?ZAL	allocate command			2-443
?ZBL	batch_list command	2-127	2-105	2-444
?ZBO	batch_output command	2-128	2-106	2-445
?ZBI	binary command	2-129	2-107	2-446
?ZBR	brief command	2-130	2-108	2-447
?ZCA	cancel command	2-131	2-109	2-448
?ZCL	close command			2-448
?ZCS	consolestatus command	2-132	2-110	2-449
?ZCO	continue command	2-133	2-111	2-451
?ZCP	cpl command	2-134	2-112	2-452
?ZCR	create command	2-135	2-113	2-453
?ZDF	defaultforms command	2-136	2-115	2-454
?ZDE	delete command			2-454
?ZDI	disable command	2-137	2-116	2-455
?ZDM	mdump command			2-471
?ZDS	dismounted command	2-138	2-117	2-456
?ZEL	elongate command	2-139	2-118	2-457
?ZEN	enable command	2-140	2-119	2-458
?ZEV	even command	2-141	2-120	2-460
?ZFL	flush command	2-142	2-121	2-461
?ZFN	font command			2-461
?ZFO	forms command	2-143	2-122	2-462
?ZHA	halt command	2-144	2-123	2-463
?ZHE	headers command	2-145	2-124	2-464
?ZHO	hold command	2-146	2-125	2-465
?ZLI	limit command	2-147	2-126	2-466
?ZLO	logging command	2-148	2-127	2-467
?ZLP	lpp command	2-149	2-128	2-469
?ZMP	mapper command	2-150	2-129	2-470
?ZME	message command			2-471
?ZMD	modify command			2-471
?ZMO	mounted command	2-151	2-130	2-472
?ZMS	mountstatus command	2-152	2-131	2-473
?ZON	open command			2-476
?ZOP	operator command	2-153	2-132	2-476
?ZPA	pause command	2-154	2-133	2-477

(continued)

Table 2–102. Contents of ?OPEX Main Packet

Offset	Contents				
?ZXFU (cont.)	Value	Functional Description	See Figure	See Table	See Page
	?ZPE	premount	2-155	2-134	2-478
	?ZPI	priority	2-156	2-135	2-479
	?ZPR	prompts command	2-157	2-136	2-480
	?ZPU	purge command			2-480
	?ZQP	queuepriority command	2-158	2-137	2-481
	?ZRF	refused command	2-159	2-138	2-483
	?ZRE	release command	2-160	2-139	2-484
	?ZRT	restart command	2-161	2-140	2-485
	?ZSI	silence command	2-162	2-141	2-486
	?ZSP	spoolstatus command	2-163	2-142	2-487
	?ZSK	stack command	2-164	2-143	2-491
	?ZSR	start command	2-165	2-144	2-492
	?ZSS	status command	2-166	2-145	2-495
	?ZST	stop command	2-167	2-146	2-499
	?ZTE	terminate command			2-499
	?ZTR	trailers command	2-168	2-147	2-500
	?ZUH	unhold command	2-169	2-148	2-501
	?ZUS	unitstatus command	2-170	2-149	2-502
	?ZUL	unlimit command	2-171	2-150	2-504
	?ZUN	unsilence command	2-172	2-151	2-505
	?XUC	user command	2-173	2-152	2-506
	?ZVE	verbose command	2-174	2-153	2-508
	?ZXB	xbias command			2-508

(continued)

Table 2–102. Contents of ?OPEX Main Packet

Offset	Contents
?ZXTP	Time prompt. This offset receives the time prompt from the operating system. If prompts are off, the operating system instead returns -1 to this offset.
?ZXFG	Active field flags. This offset tells the operating system that offset ?ZXNA contains a byte pointer to a buffer that you are using. For example, if the function you specify (such as ?ZAC -- the access command) requires placing a buffer byte pointer in offset ?ZXNA, then place the value ?ZYA0 (= 1B(?ZZA0)) in offset ?ZXFG. If you don't place ?ZYA0 in offset ?ZXFG, then place zero in offsets ?ZXNA and ?ZXNB. The function codes ?ZHA (halt command request), ?ZMD (modify command request), ?ZPR (prompts), and ?ZRF (refused) don't require a buffer byte pointer; place zero in offsets ?ZXNA and ?ZXNB for them.
?ZXRS	Reserved. (Set to 0.)
?ZXNA (double-word)	This offset contains a byte pointer to the buffer with the device name, queue name, pathname, or username of the function you are requesting.
?ZXNB	Supply the number of bytes in the buffer that offset ?ZXNA points to.
?ZXNL	The operating system returns the number of bytes in the name that you placed in offset ?ZXNA.
?ZXSP (double-word)	Supply the word address of the subpacket for the function you specified in offset ?ZXFU. If the function doesn't require a subpacket, place zero here.

(concluded)

Access Command

Offset ?ZXNA of the main ?OPEX packet in Figure 2–124 must contain the byte address of the device/queue name/cooperative buffer.

Figure 2–125 shows the structure of the access command subpacket, and Table 2–103 describes its contents.

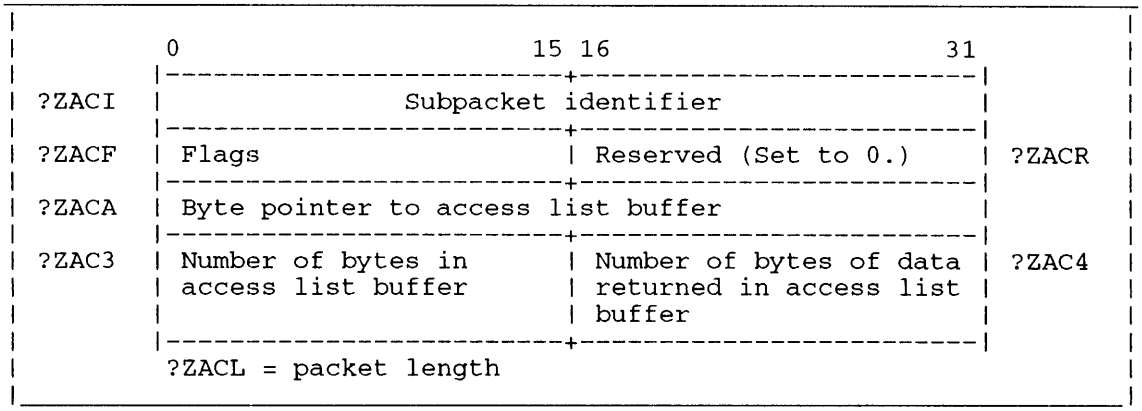


Figure 2–125. Structure of Access Command Subpacket

Table 2–103. Contents of Access Command Subpacket

Offset	Contents
?ZACI (double-word)	Subpacket identifier. Place ?ZACZ here.
?ZACF	Flags word. It contains the bit flags with additional information to the operating system and EXEC for function ?ZAC. ?ZY9G = 1B(?ZZ9G) -- default. ?ZY9H = 1B(?ZZ9H) -- specify the kill option. ?ZY9I = 1B(?ZZ9I) -- operator-specified ACL.
?ZACR	Reserved. (Set to 0.)
?ZACA (double-word)	Supply a byte pointer to your access list buffer.
?ZAC3	Specify the number of bytes in your access list buffer.
?ZAC4	The operating system returns the number of bytes in the access list that is in your access list buffer.

Align Command

Offset ?ZXNA of the main ?OPEX packet in Figure 2–124 must contain the byte address of the device name buffer for the align command.

Figure 2–126 shows the structure of the align command subpacket, and Table 2–104 describes its contents.

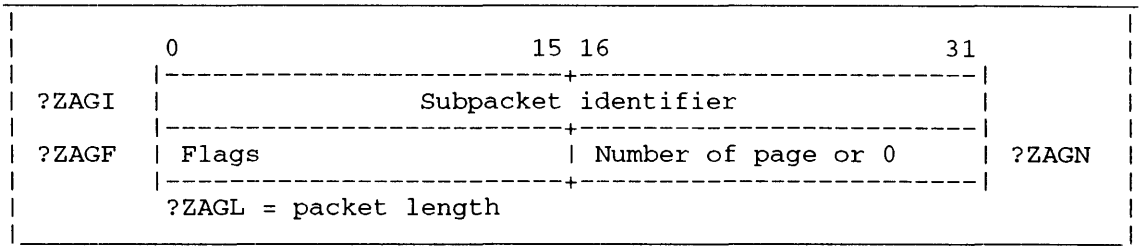


Figure 2–126. Structure of Align Command Subpacket

Table 2–104. Contents of Align Command Subpacket

Offset	Contents
?ZAGI (double-word)	Subpacket identifier. Place ?ZAGZ here.
?ZAGF	Flags word. It contains the bit flags with additional information to the operating system and EXEC for function ?ZAG. ?ZYE0 = 1B(?ZZE0) -- continue.
?ZAGN	Supply the page number. Or, if you supply the default page number of zero, then continuation occurs where utility program XLPT left off.

Allocate Command

Offset ?ZXNA of the main ?OPEX packet in Figure 2–124 must contain the byte address of the device name buffer.

The allocate function does not require a subpacket.

Batch_List Command

Offset ?ZXNA of the main ?OPEX packet in Figure 2–124 must contain the byte address of the batch queue name buffer for the batch_list function.

Figure 2–127 shows the structure of the batch_list command subpacket, and Table 2–105 describes its contents.

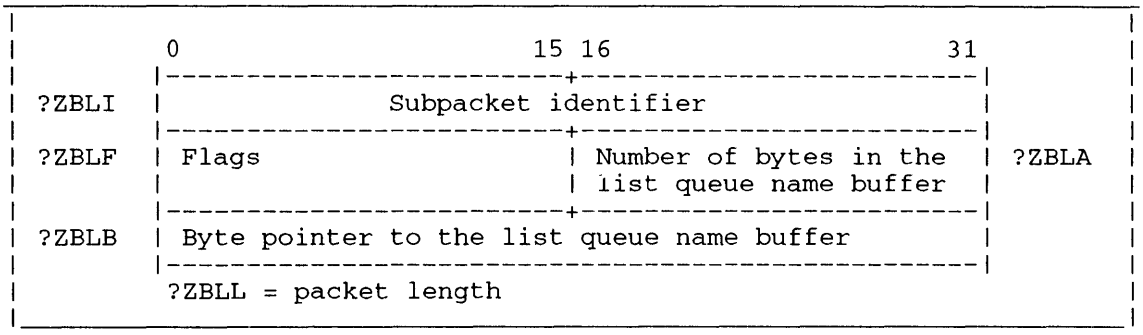


Figure 2–127. Structure of Batch_List Command Subpacket

Table 2–105. Contents of Batch_List Command Subpacket

Offset	Contents
?ZBLI (double-word)	Subpacket identifier. Place ?ZBLZ here.
?ZBLF	Flags word. It contains the bit flags with additional information to the operating system and EXEC for function ?ZBL. ?ZYL0 = 1B(?ZZL0) -- The default switch is on.
?ZBLA	Specify the number of bytes in the list queue name buffer.
?ZBLB (double-word)	Supply a byte pointer to the list queue name buffer.

Batch_Output Command

Offset ?ZXNA of the main ?OPEX packet in Figure 2–124 must contain the byte address of the batch queue name buffer for the batch_output function.

Figure 2–128 shows the structure of the batch_output command subpacket, and Table 2–106 describes its contents.

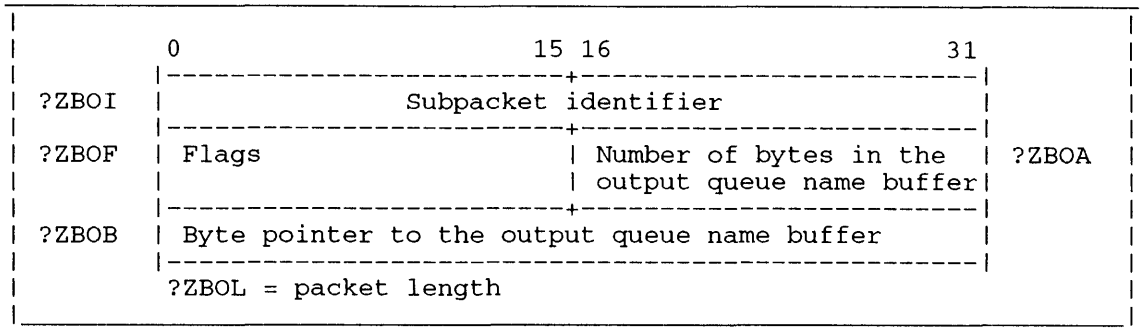


Figure 2–128. Structure of Batch_Output Command Subpacket

Table 2–106. Contents of Batch_Output Command Subpacket

Offset	Contents
?ZBOI (double-word)	Subpacket identifier. Place ?ZBOZ here.
?ZBOF	Flags word. It contains the bit flags with additional information to the operating system and EXEC for function ?ZBO. ?ZYM0 = 1B(?ZZM0) -- The default switch is on.
?ZBOA	Specify the number of bytes in the output queue name buffer.
?ZBOB (double-word)	Supply a byte pointer to the output queue name buffer.

Binary Command

Offset ?ZXNA of the main ?OPEX packet in Figure 2–124 must contain the byte address of the device name buffer for the binary function.

Figure 2–129 shows the structure of the binary command subpacket, and Table 2–107 describes its contents.

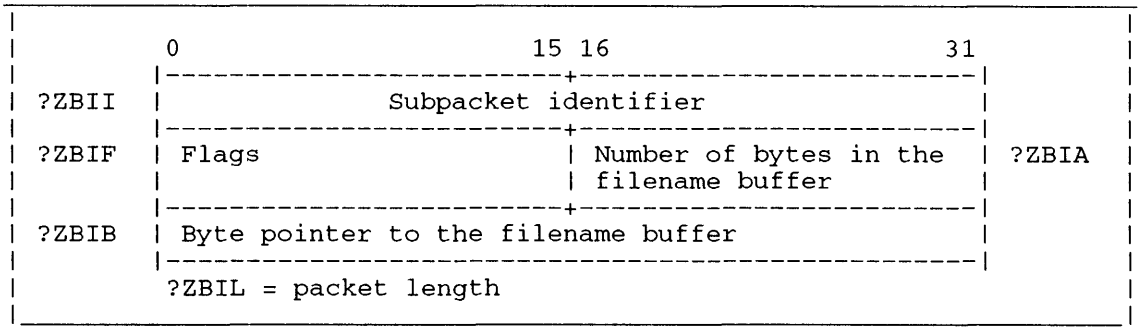


Figure 2–129. Structure of Binary Command Subpacket

Table 2–107. Contents of Binary Command Subpacket

Offset	Contents
?ZBII (double-word)	Subpacket identifier. Place ?ZBIZ here.
?ZBIF	Flags word. It contains the bit flags with additional information to the operating system and EXEC for function ?ZBI. ?ZYF0 = 1B(?ZZF0) -- The byte pointer to the filename buffer is passed in offset ?ZBIB. ?ZYF1 = 1B(?ZZF1) -- Sets binary mode to OFF. ?ZJF1 = 0B(?ZZF1) -- Sets binary mode to ON. ?ZYF2 = 1B(?ZZF2) -- Return value that specifies the binary mode is enabled. ?ZYJ2 = 0B(?ZZF2) -- Return value that specifies the binary mode is disabled.
?ZBIA	Specify the number of bytes in the filename buffer.
?ZBIB (double-word)	Supply a byte pointer to the filename buffer. Place a filename in this buffer.

Brief Command

Offset ?ZXNA of the main ?OPEX packet in Figure 2–124 must contain the byte address of the device/queue name buffer for the brief function. If you set flag bit ?ZZN0 in offset ?ZBRF of the subpacket, then the request is for all batch streams.

Figure 2–130 shows the structure of the brief command subpacket, and Table 2–108 describes its contents.

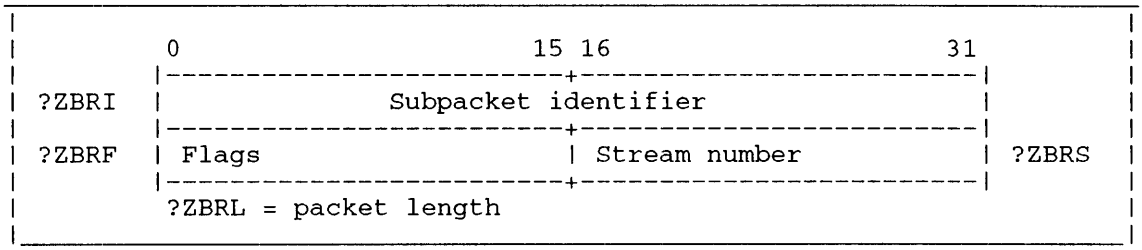


Figure 2–130. Structure of Brief Command Subpacket

Table 2–108. Contents of Brief Command Subpacket

Offset	Contents
?ZBRI (double-word)	Subpacket identifier. Place ?ZBRZ here.
?ZBRF	Flags word. It contains the bit flags with additional information to the operating system and EXEC for function ?ZBR. ?ZYN0 = 1B(?ZZN0) -- All default batch input streams are specified.
?ZBRS	Supply the batch stream number if bit ?ZZN0 in offset ?ZBRF is not set. Otherwise, supply zero (for all batch streams).

Cancel Command

Offset ?ZXNA of the main ?OPEX packet in Figure 2–124 must contain the byte address of the username buffer, if necessary, for the cancel function.

Figure 2–131 shows the structure of the cancel command subpacket, and Table 2–109 describes its contents.

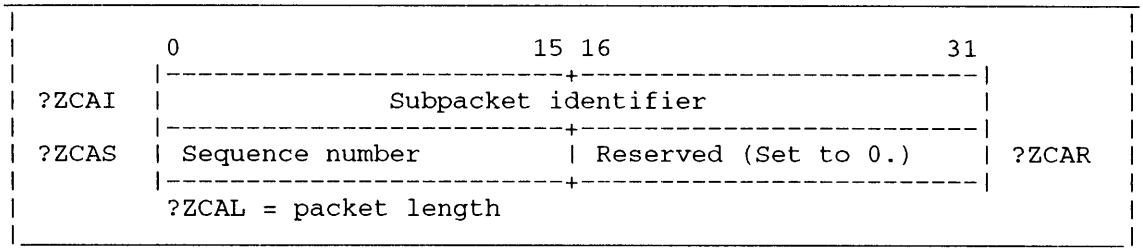


Figure 2–131. Structure of Cancel Command Subpacket

Table 2–109. Contents of Cancel Command Subpacket

Offset	Contents
?ZCAI (double-word)	Subpacket identifier. Place ?ZCAZ here.
?ZCAS	Supply the sequence number in this offset. Or, to cancel all jobs with a given username, supply 0 here and supply a byte pointer to the username in offset ?ZXNA of the main packet of ?OPEX.
?ZCAR	Reserved (Set to 0.)

Close Command

Offset ?ZXNA of the main ?OPEX packet in Figure 2–124 must contain a byte pointer to the queue name buffer that contains the queue name for the close function.

The close function does not require a subpacket.

Consolestatus Command

Offset ?ZXNA of the main ?OPEX packet in Figure 2–124 must contain the byte address of the device name buffer for the consolestatus function. If you take the default (by setting bit ?ZZB0 of offset ?ZCSF), then the operating system returns the name of the terminal in the device name buffer.

Figure 2–132 shows the structure of the consolestatus command subpacket, and Table 2–110 describes its contents.

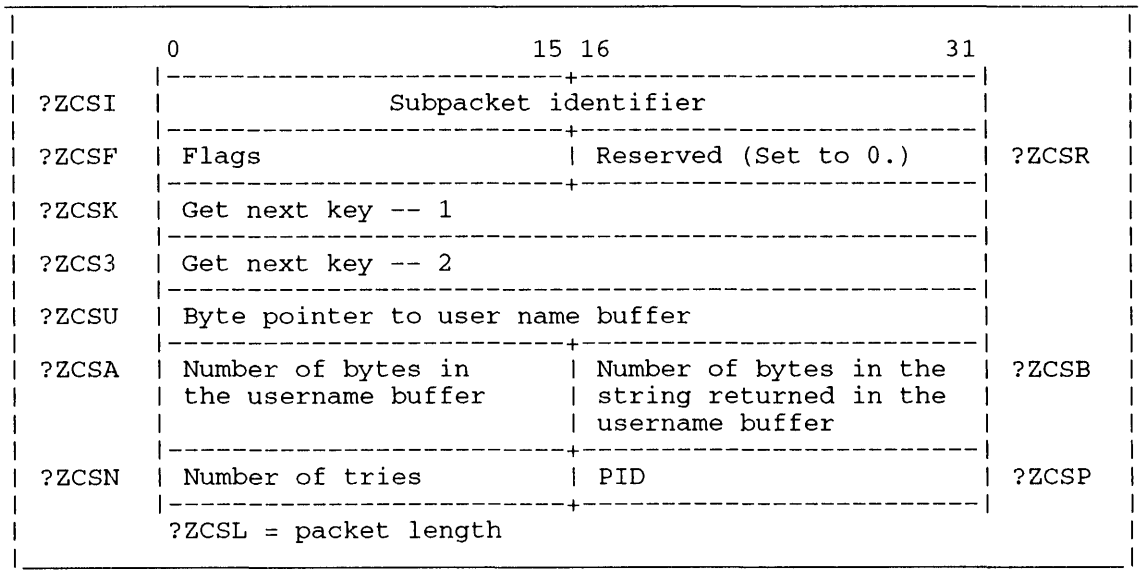


Figure 2–132. Structure of Consolestatus Command Subpacket

Table 2–110. Contents of Consolestatus Command Subpacket

Offset	Contents
?ZCSI (double-word)	Subpacket identifier. Place ?ZCSZ here.
?ZCSF	Flags word. It contains the bit flags with additional information to the operating system and EXEC for function ?ZCS. ?ZYB0 = 1B(?ZZB0) -- The default. You want the status of all terminals. ?ZYB1 = 1B(?ZZB1) -- Terminal is enabled. ?ZYB2 = 1B(?ZZB2) -- Continue. ?ZYB3 = 1B(?ZZB3) -- Stop. ?ZYB4 = 1B(?ZZB4) -- Terminal is logged on. ?ZYB5 = 1B(?ZZB5) -- Terminal will be disabled. ?ZYB6 = 1B(?ZZB6) -- Log on/log off is in progress. ?ZYB7 = 1B(?ZZB7) -- Terminal is in server died state. ?ZYB8 = 1B(?ZZB8) -- Terminal is in error state. ?ZYB9 = 1B(?ZZB9) -- Terminal is not logged on.
?ZCSR	Reserved. (Set to 0.)
?ZCSK (double-word)	Get next entry key. The operating system uses this offset to hold the key indicator for EXEC.
?ZCS3 (double-word)	Second get next entry key. The operating system uses this offset to hold the second key indicator for EXEC.
?ZCSU (double-word)	Supply a byte pointer to the buffer that receives the username.
?ZCSA	Specify a number of bytes to accommodate the largest possible username.
?ZCSB	The operating system returns the number of bytes in the username that it placed in the username buffer.
?ZCSN	The operating system returns the number of tries it made.
?ZCSP	The operating system returns the PID associated with the terminal.

Continue Command

Offset ?ZXNA of the main ?OPEX packet in Figure 2–124 must contain the byte address of the device/queue name buffer for the continue function.

Figure 2–133 shows the structure of the continue command subpacket, and Table 2–111 describes its contents.

If you want to continue all default batch input streams, follow these steps:

1. Set Bit ?ZZOO in offset ?ZCOF on.
2. Set the remaining bits in offset ?ZCOF off.
3. Place zero in offset ?ZCOS.
4. In the main packet, set the following to zero:
   - Offset ?ZXNA
   - Offset ?ZXNB
   - Bit ?ZZA0 in offset ?ZXFG

Suppose you want to continue all default streams associated with a batch queue you created via the ?ZCR function of ?OPEX. You have also specified the name of the queue and type=batch. Then, place zero in offsets ?ZCOF and ?ZCOS. In the main packet the offsets ?ZXNA, ?ZXNB, and ?ZXFG do not contain zero.

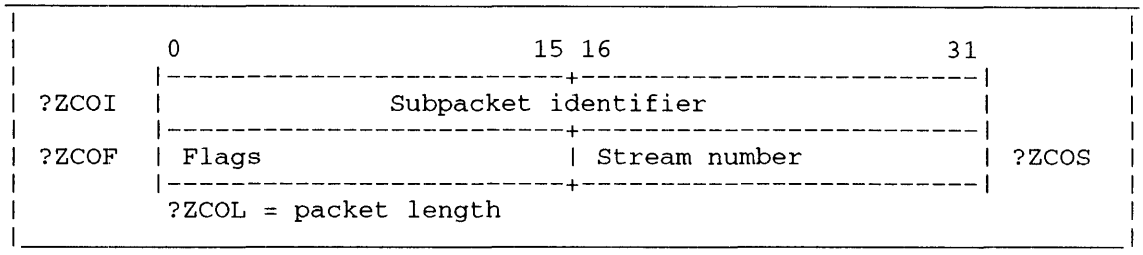


Figure 2–133. Structure of Continue Command Subpacket

Table 2–111. Contents of Continue Command Subpacket

Offset	Contents
?ZCOI (double-word)	Subpacket identifier. Place ?ZCOZ here.
?ZCOF	Flags word. It contains bit flags that indicate default choices to the operating system. ?ZYO0 = 1B(?ZZO0) -- Continue all default batch input streams.
?ZCOS	Supply the stream number. The default (the value 0) is to continue all batch streams.

CPL (Characters Per Line) Command

Offset ?ZXNA of the main ?OPEX packet in Figure 2–124 must contain the byte address of the device name buffer for the cpl function.

Figure 2–134 shows the structure of the cpl command subpacket, and Table 2–112 describes its contents.

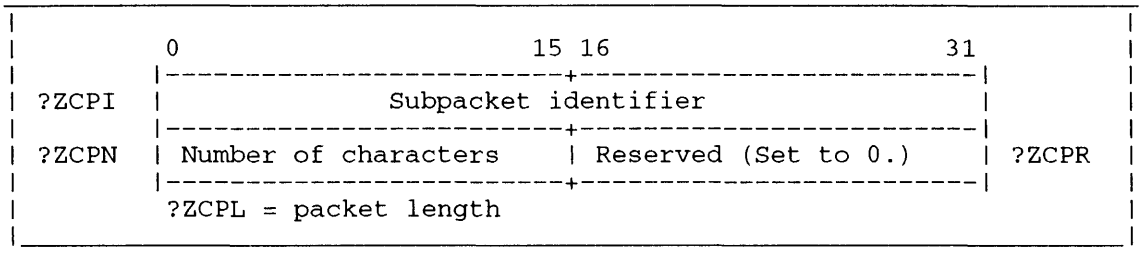


Figure 2–134. Structure of CPL Command Subpacket

Table 2–112. Contents of CPL Command Subpacket

Offset	Contents
?ZCPI (double-word)	Subpacket identifier. Place ?ZCPZ here.
?ZCPN	Supply the number of characters per line.
?ZCPR	Reserved. (Set to 0.)

Create Command

Offset ?ZXNA of the main ?OPEX packet in Figure 2–124 must contain the byte address of the queue name buffer for the create function.

Figure 2–135 shows the structure of the create command subpacket, and Table 2–113 describes its contents.

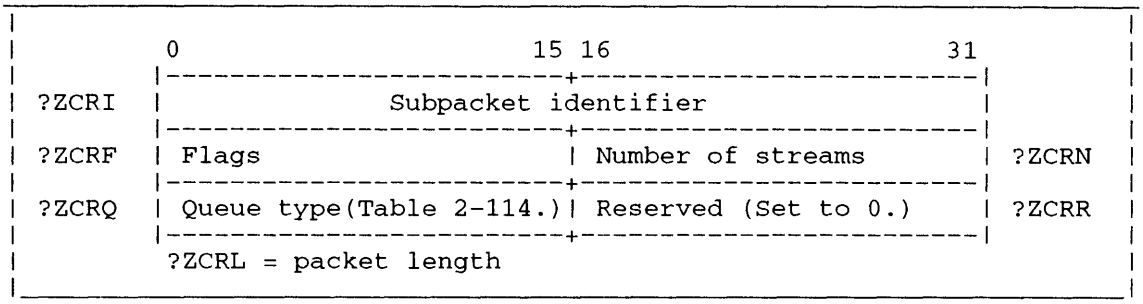


Figure 2–135. Structure of Create Command Subpacket

Table 2–113. Contents of Create Command Subpacket

Offset	Contents
?ZCRI (double-word)	Subpacket identifier. Place ?ZCRZ here.
?ZCRF	Flags word. It contains the bit flags with additional information to the operating system and EXEC for function ?ZCR. ?ZYP0 = 1B(?ZZP0) -- Streams switch.
?ZCRN	Supply the number of streams.
?ZCRQ	Supply the numeric queue type.
?ZCRR	Reserved. (Set to 0.)

Table 2–114 lists the queue types that EXEC supports.

Table 2–114. ?OPEX Queue Types in Offset ?ZCRQ

Offset Value	Queue Type	Process Type
?XQBAT	BATCH	Batch processing
?XQLPT	PRINT	Print processing
?XQPLT	PLOT	Plot processing
?XQHAM	HAMLET	HAMLET data transfer
?XQSNA	SNA	SNA data transfer
?XQFTA	FTA	FTA data transfer
?XQMUN	MOUNT	Mount processing
?XQUSR	USER	User-defined processing

Defaultforms Command

Offset ?ZXNA of the main ?OPEX packet in Figure 2–124 must contain the byte address of the device name buffer for the defaultforms function.

Figure 2–136 shows the structure of the defaultforms command subpacket, and Table 2–115 describes its contents.

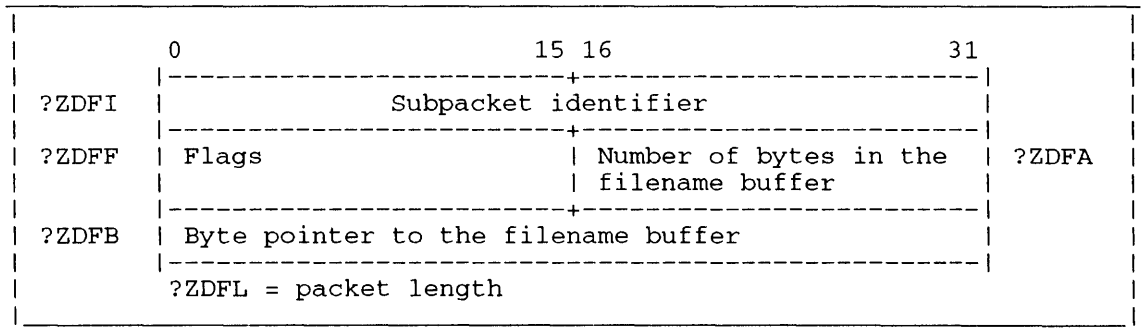


Figure 2–136. Structure of Defaultforms Command Subpacket

Table 2–115. Contents of Defaultforms Command Subpacket

Offset	Contents
?ZDFI (double-word)	Subpacket identifier. Place ?ZDFZ here.
?ZDFF	Flags word. It contains the bit flags that indicate default choices to the operating system. ?ZYG0 = 1B(?ZZG0) -- If the bit is set, then the byte pointer to the filename buffer is in offset ?ZDFB. Otherwise, the default forms are used.
?ZDFA	Specify the number of bytes in the filename buffer.
?ZDFB (double-word)	Supply a byte pointer to the filename buffer. Place a filename in this buffer.

Delete Command

Offset ?ZXNA of the main ?OPEX packet in Figure 2–124 must contain a byte pointer to the queue name buffer that contains the queue name for the delete function.

The delete function does not require a subpacket.

Disable Command

Offset ?ZXNA of the main ?OPEX packet in Figure 2–124 must contain the byte address of the device name buffer for the disable function.

Figure 2–137 shows the structure of the disable command subpacket, and Table 2–116 describes its contents.

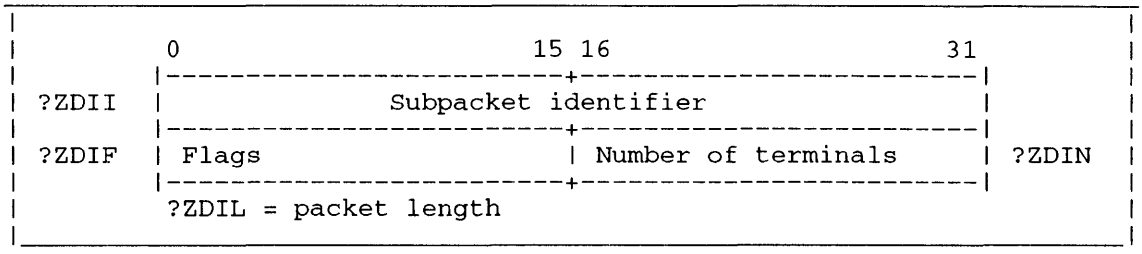


Figure 2–137. Structure of Disable Command Subpacket

Table 2–116. Contents of Disable Command Subpacket

Offset	Contents
?ZDII (double-word)	Subpacket identifier. Place ?ZDIZ here.
?ZDIF	Flags word. It contains the bit flags with additional information to the operating system and EXEC for function ?ZDI. ?ZYC0 = 1B(?ZZC0) -- Default; you want to disable all the terminals. ?ZYC1 = 1B(?ZZC1) -- The terminal will be disabled. ?ZYC2 = 1B(?ZZC2) -- All terminals will be disabled.
?ZDIN	Number of terminals. Supply 0. If you set bit ?ZZC0 of offset ?ZDIF, AOS/VS returns the number of terminals disabled in offset ?ZDIN.

Dismounted Command

Offset ?ZXNA of the main ?OPEX packet in Figure 2–124 must contain the byte address of the device name buffer for the dismounted function.

Figure 2–138 shows the structure of the dismounted command subpacket, and Table 2–117 describes its contents.

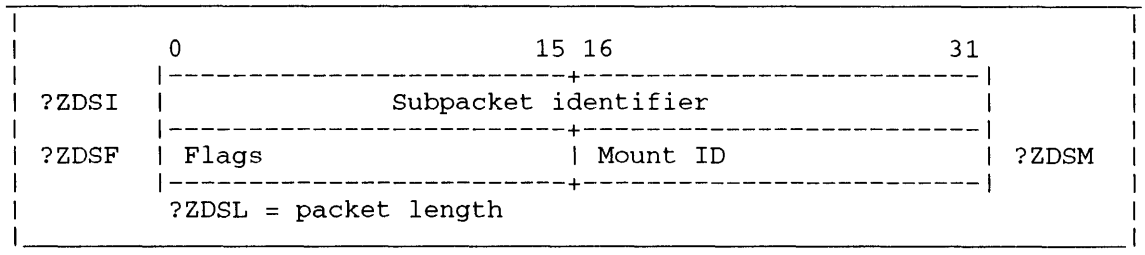


Figure 2–138. Structure of Dismounted Command Subpacket

Table 2–117. Contents of Dismounted Command Subpacket

Offset	Contents
?ZDSI (double-word)	Subpacket identifier. Place ?ZDSZ here.
?ZDSF	Flags word. It contains the bit flags that indicate default choices to the operating system. ?ZY00 = 1B(?ZZ00) -- Default mount request.
?ZDSM	Supply the mount ID.

Elongate Command

Offset ?ZXNA of the main ?OPEX packet in Figure 2–124 must contain the byte address of the device name buffer for the elongate function.

Figure 2–139 shows the structure of the binary command subpacket, and Table 2–118 describes its contents.

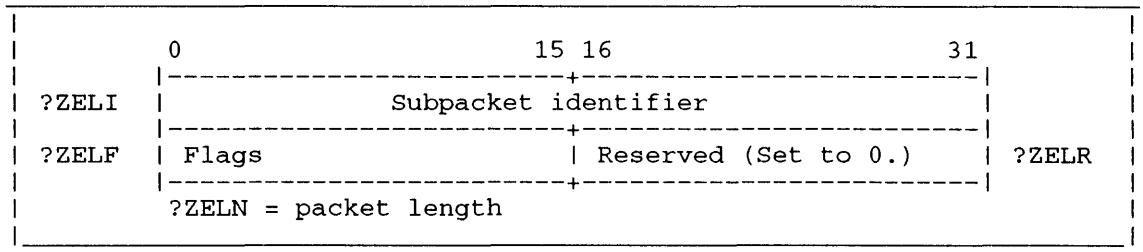


Figure 2–139. Structure of Elongate Command Subpacket

Table 2–118. Contents of Elongate Command Subpacket

Offset	Contents
?ZELI (double-word)	Subpacket identifier. Place ?ZELZ here.
?ZELF	Flags word. It contains the bit flags with additional information to the operating system and EXEC for function ?ZEL. ?ZYH0 = 1B(?ZZH0) -- Elongate is turned on. ?ZJH0 = 0B(?ZZH0) -- Elongate is turned off.
?ZELR	Reserved. (Set to 0.)

Enable Command

Offset ?ZXNA of the main ?OPEX packet in Figure 2–124 must contain a byte pointer to the device name buffer for the enable function. The operating system will return the terminal name if you specify so.

Figure 2–140 shows the structure of the enable command subpacket, and Table 2–119 describes its contents.

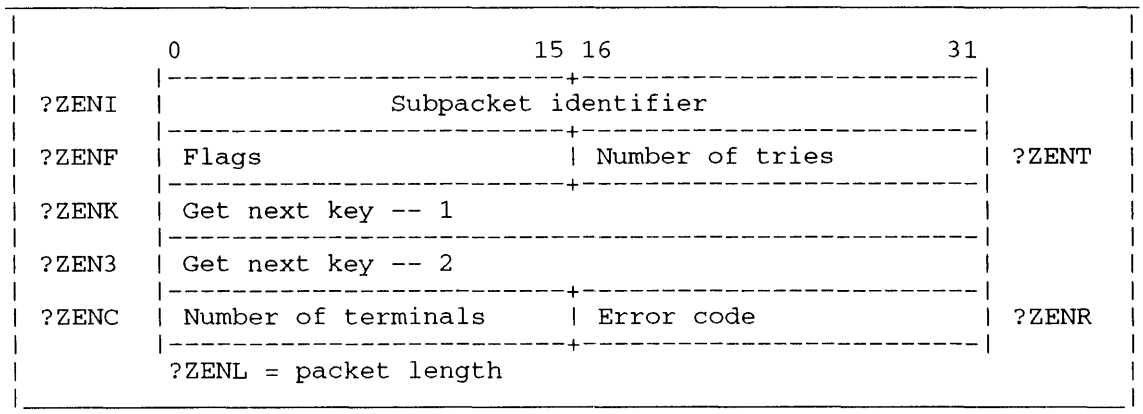


Figure 2–140. Structure of Enable Command Subpacket

Table 2–119. Contents of Enable Command Subpacket

Offset	Contents
?ZENI (double-word)	Subpacket identifier. Place ?ZENZ here.
?ZENF	Flags word. It contains the bit flags with additional information to the operating system and EXEC for function ?ZEN. ?ZYD0 = 1B(?ZZD0) -- The default. You want to enable all terminals. ?ZYD1 = 1B(?ZZD1) -- Tries switch. ?ZYD2 = 1B(?ZZD2) -- Stop switch. ?ZYD3 = 1B(?ZZD3) -- Continue switch. ?ZYD4 = 1B(?ZZD4) -- Force switch. ?ZYD5 = 1B(?ZZD5) -- Terminal is already enabled. ?ZYD6 = 1B(?ZZD6) -- Cancelling disable, previous values are in effect. ?ZYD7 = 1B(?ZZD7) -- Cancelling disable, new log-on values are in effect. ?ZYD8 = 1B(?ZZD8) -- New log-on values are in effect. ?ZYD9 = 1B(?ZZD9) -- Terminal enabled. ?ZYDA = 1B(?ZZDA) -- Could not enable terminal. ?ZYDB = 1B(?ZZDB) -- Device already in use.
?ZENT	Supply (to EXEC) the maximum number of log-on tries.
?ZENK (double-word)	Get next entry key. The operating system uses this offset to hold the key indicator for EXEC.
?ZEN3 (double-word)	Second get next entry key. The operating system uses this offset to hold the second key indicator for EXEC.
?ZENC	The operating system returns the number of terminals enabled when you set the all terminals value (?ZYD0) in offset ?ZENF.
?ZENR	The operating system returns, from EXEC, an error code.

Even Command

Offset ?ZXNA of the main ?OPEX packet in Figure 2–124 must contain the byte address of the device name buffer for the even function.

Figure 2–141 shows the structure of the even command subpacket, and Table 2–120 describes its contents.

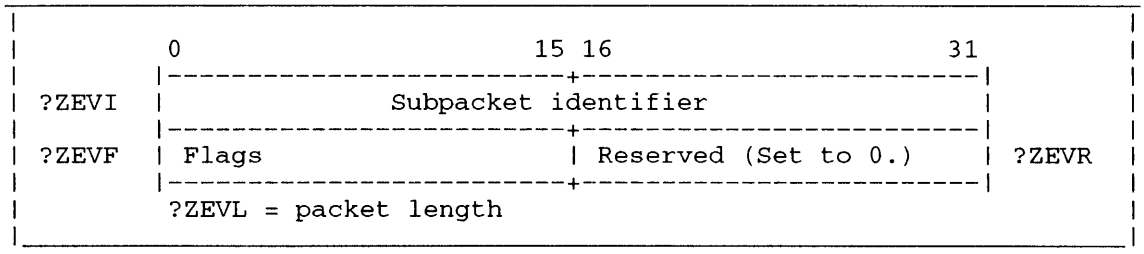


Figure 2–141. Structure of Even Command Subpacket

Table 2–120. Contents of Even Command Subpacket

Offset	Contents
?ZEVI (double-word)	Subpacket identifier. Place ?ZEVZ here.
?ZEVF	Flags word. It contains the bit flags with additional information to the operating system and EXEC for the function ?ZEV. ?ZYI0 = 1B(?ZZI0) -- Even is turned on. ?ZJI0 = 0B(?ZZI0) -- Even is turned off.
?ZEVR	Reserved. (Set to 0.)

Flush Command

Offset ?ZXNA of the main ?OPEX packet in Figure 2–124 must contain the byte address of the device/queue name buffer for the flush function.

Figure 2–142 shows the structure of the flush command subpacket, and Table 2–121 describes its contents.

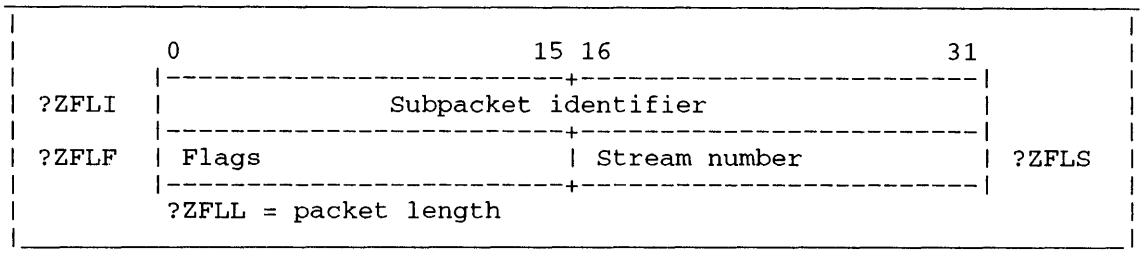


Figure 2–142. Structure of Flush Command Subpacket

Table 2–121. Contents of Flush Command Subpacket

Offset	Contents
?ZFLI (double-word)	Subpacket identifier. Place ?ZFLZ here.
?ZFLF	Flags word. Set to 0.
?ZFLS	Supply the stream number. If the target is a device with no streams, supply 0.

Font Command

This function is currently undefined.

Forms Command

Offset ?ZXNA of the main ?OPEX packet in Figure 2–124 must contain the byte address of the device name buffer for the forms function.

Figure 2–143 shows the structure of the forms command subpacket, and Table 2–122 describes its contents.

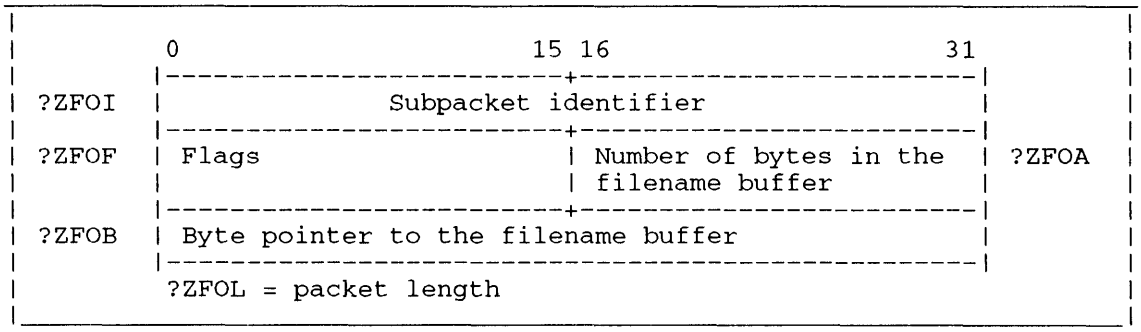


Figure 2–143. Structure of Forms Command Subpacket

Table 2–122. Contents of Forms Command Subpacket

Offset	Contents
?ZFOI (double-word)	Subpacket identifier. Place ?ZFOZ here.
?ZFOF	Flags word. It contains the bit flags that indicate default choices to the operating system. ?ZYJ0 = 1B(?ZZJ0) -- If the bit is set, then the byte pointer to the filename buffer is in offset ?ZFOB. Otherwise, the implicit default forms are used.
?ZFOA	Specify the number of bytes in the filename buffer.
?ZFOB (double-word)	Supply a byte pointer to the filename buffer. Place a filename in this buffer.

Halt Command

Bit ?ZZA0 of offset ?ZXFG of the main ?OPEX packet in Figure 2–124 must not be set, since the halt function does not require a buffer to pass information. Consequently, the operating system ignores offset ?ZXNA of the main ?OPEX packet.

Figure 2–144 shows the structure of the halt command subpacket, and Table 2–123 describes its contents.

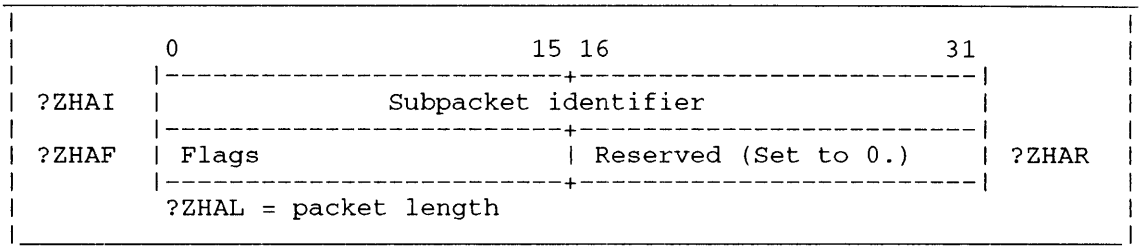


Figure 2–144. Structure of Halt Command Subpacket

Table 2–123. Contents of Halt Command Subpacket

Offset	Contents
?ZHAI (double-word)	Subpacket identifier. Place ?ZHAZ here.
?ZHAF	Flags word. There are no bits defined in this word. Set it to 0.
?ZHAR	Reserved. (Set to 0.)

Headers Command

Offset ?ZXNA of the main ?OPEX packet in Figure 2–124 must contain the byte address of the device name buffer for the headers function.

Figure 2–145 shows the structure of the headers command subpacket, and Table 2–124 describes its contents.

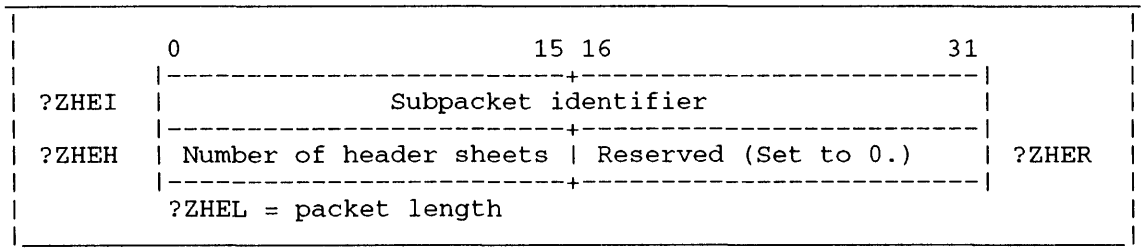


Figure 2–145. Structure of Headers Command Subpacket

Table 2–124. Contents of Headers Command Subpacket

Offset	Contents
?ZHEI (double-word)	Subpacket identifier. Place ?ZHEZ here.
?ZHEH	Supply the number of header sheets before each printed job.
?ZHER	Reserved. (Set to 0.)

Hold Command

Offset ?ZXNA of the main ?OPEX packet in Figure 2–124 must contain the byte address of the username buffer, if necessary, for the hold function.

Figure 2–146 shows the structure of the hold command subpacket, and Table 2–125 describes its contents.

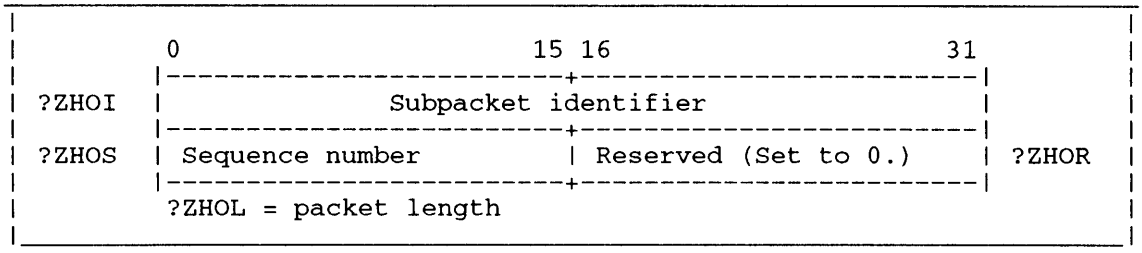


Figure 2–146. Structure of Hold Command Subpacket

Table 2–125. Contents of Hold Command Subpacket

Offset	Contents
?ZHOI (double-word)	Subpacket identifier. Place ?ZHOZ here.
?ZHOS	Supply the sequence number in this offset. Or, to hold all jobs with a given username, supply 0 here and supply a byte pointer to the username in offset ?ZXNA of the main packet of ?OPEX.
?ZHOR	Reserved. (Set to 0.)

Limit Command

Offset ?ZXNA of the main ?OPEX packet in Figure 2–124 must contain the byte address of the device/queue name buffer for the limit function.

There are two correspondences between the CLI syntax for the limit command and the ?OPEX equivalent. They are next.

```
CONTROL @EXEC LIMIT  <QUEUE NAME>          [STREAM #]       [HH:MM:SS]
?OPEX         FUNC   (OPEX_PACKET.NAME)    ARGUMENT_1       ARGUMENT_2

CONTROL @EXEC LIMIT  @<DEVICENAME>         [MAX]
?OPEX         FUNC   (OPEX_PACKET.NAME)    ARGUMENT_2
```

Figure 2–147 shows the structure of the limit command subpacket, and Table 2–126 describes its contents.

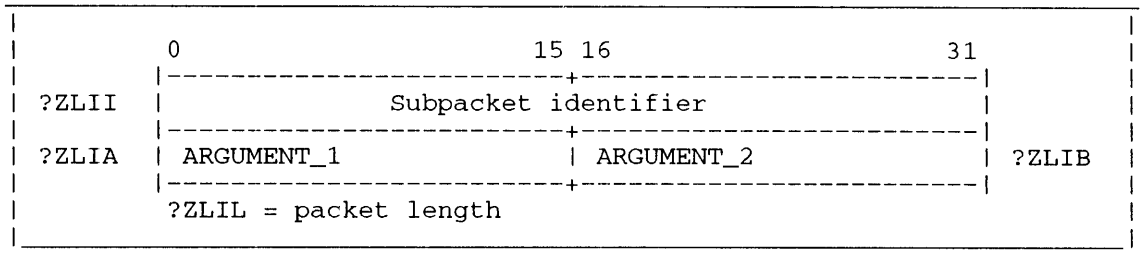


Figure 2–147. Structure of Limit Command Subpacket

Table 2–126. Contents of Limit Command Subpacket

Offset	Contents
?ZLII (double-word)	Subpacket identifier. Place ?ZLIZ here.
?ZLIA	ARGUMENT_1. If you specify a queue name, supply a stream number here. The default value is all batch streams; to specify this, supply -1. If you specify a device name, then supply -1.
?ZLIB	ARGUMENT_2. If you specify a queue name, supply the time. The default time is 36:24:30; to specify this, supply -1. If you specify a device name, then supply the maximum number of pages. The default value of -1 specifies 65535 as the maximum number of pages.

Logging Command

Offset ?ZXNA of the main ?OPEX packet in Figure 2–124 must contain the byte address of the pathname or device name buffer for the logging function. For logging status requests, this buffer will receive the current log file name from the operating system.

Figure 2–148 shows the structure of the logging command subpacket, and Table 2–127 describes its contents.

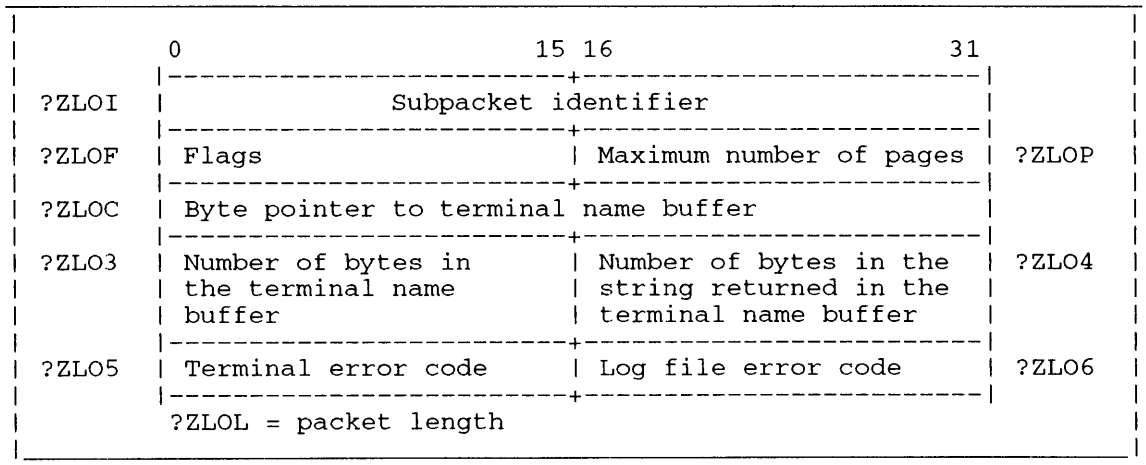


Figure 2–148. Structure of Logging Command Subpacket

Table 2–127. Contents of Logging Command Subpacket

Offset	Contents
?ZLOI (double-word)	Subpacket identifier. Place ?ZLOZ here.
?ZLOF	Flags word. It contains the bit flags to indicate default choices to the operating system. ?ZY80 = 1B(?ZZ80) -- Return status. This is the default. ?ZY81 = 1B(?ZZ81) -- Start switch. ?ZY82 = 1B(?ZZ82) -- Maximum switch. ?ZY83 = 1B(?ZZ83) -- Stop switch. ?ZY84 = 1B(?ZZ84) -- Terminal switch. ?ZY85 = 1B(?ZZ85) -- No terminal switch. ?ZY86 = 1B(?ZZ86) -- Start logging. ?ZY87 = 1B(?ZZ87) -- Stop logging.
?ZLOP	Supply the maximum number of pages.
?ZLOC (double-word)	Supply a byte pointer to the buffer that receives the terminal name.
?ZLO3	Specify a number of bytes to accommodate the largest possible terminal name.
?ZLO4	The operating system returns the number of bytes in the terminal name that it placed in the terminal name buffer.
?ZLO5	EXEC returns a terminal error code to this offset.
?ZLO6	EXEC returns a log file error code to this offset.

LPP (Lines Per Page) Command

Offset ?ZXNA of the main ?OPEX packet in Figure 2–124 must contain the byte address of the device name buffer for the lpp function.

Figure 2–149 shows the structure of the lpp command subpacket, and Table 2–128 describes its contents.

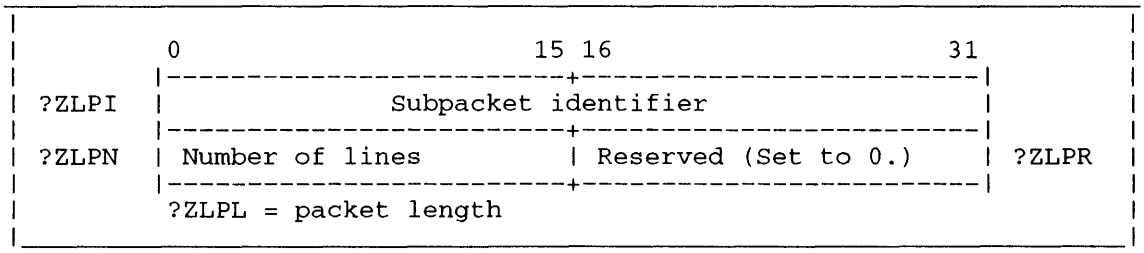


Figure 2–149. Structure of LPP Command Subpacket

Table 2–128. Contents of LPP Command Subpacket

Offset	Contents
?ZLPI (double-word)	Subpacket identifier. Place ?ZLPZ here.
?ZLPN	Supply the number of lines per page.
?ZLPR	Reserved. (Set to 0.)

Mapper Command

Offset ?ZXNA of the main ?OPEX packet in Figure 2–124 must contain the byte address of the device name buffer for the mapper function.

Figure 2–150 shows the structure of the mapper command subpacket, and Table 2–129 describes its contents.

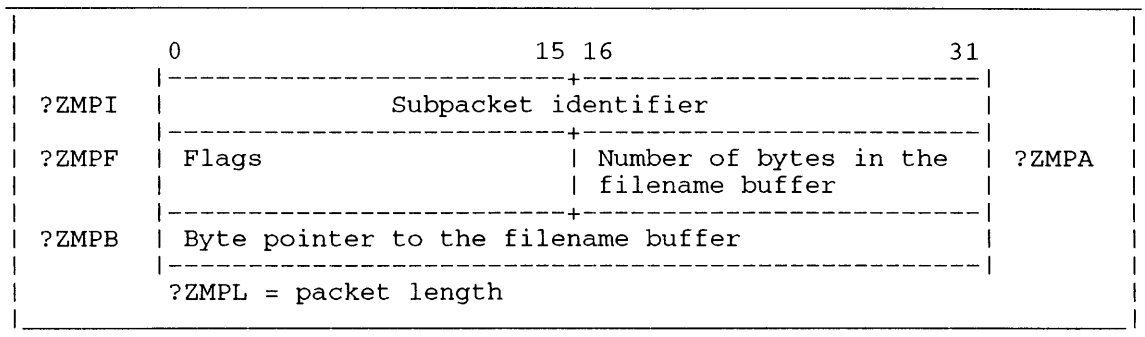


Figure 2–150. Structure of Mapper Command Subpacket

Table 2–129. Contents of Mapper Command Subpacket

Offset	Contents
?ZMPI (double-word)	Subpacket identifier. Place ?ZMPZ here.
?ZMPF	Flags word. It contains the bit flags that indicate default choices to the operating system. ?ZYMF = 1B(?ZZMF) -- If the bit is set, then the byte pointer to the mapper filename buffer is in offset ?ZMPB. Otherwise, no mapper filename is used.
?ZMPA	Specify the number of bytes in the mapper filename buffer.
?ZMPB (double-word)	Supply a byte pointer to the mapper filename buffer. Place the name of a mapper file in this buffer.

Mdump Command

This function is a request to EXEC to create a memory dump file of itself. EXEC responds by issuing system call ?MDUMP. The result is a file named ?EXEC.dd_mmm_yy.hh_mm_ss.7.MDM. As you can see, the filename is based on the date and time that ?MDUMP created the memory dump file from ring 7. ?MDUMP places the file in the same directory as the executing EXEC.PR file — typically, :UTIL. The file's ACL is the same as EXEC's default ACL — typically, OP,OWARE.

Another response to your request is a pair of messages that system call ?SEND sends to your process. These messages state that a memory dump is occurring and give the complete pathname of the memory dump file. Or, if ?MDUMP fails, your process receives a pair of messages about the failure via system call ?SEND.

This function does not require a subpacket or a buffer. Supply the following values to the main ?OPEX packet in Figure 2–124.

?ZXID	?ZXIZ
?ZXFU	?ZDM
?ZXTP	See Table 2–102.
?ZXFG	0
?ZXRS	0
?ZXNA	0
?ZXNB	0
?ZXNL	0
?ZXSP	0

Message Command

This function does not require a subpacket. Instead, supply a byte pointer to the message text buffer in offset ?ZXNA of the ?OPEX main packet in Figure 2–124.

Modify Command

Supply 0 in offset ?ZXNA of the ?OPEX main packet in Figure 2–124, since all information is passed in the subpacket. The subpacket has the same format as the one for the ?XFMOD function of system call ?EXEC. See Figure 2–159.

Mounted Command

Offset ?ZXNA of the main ?OPEX packet in Figure 2–124 must contain the byte address of the device name buffer, if necessary, for the mounted function.

Figure 2–151 shows the structure of the mounted command subpacket, and Table 2–130 describes its contents.

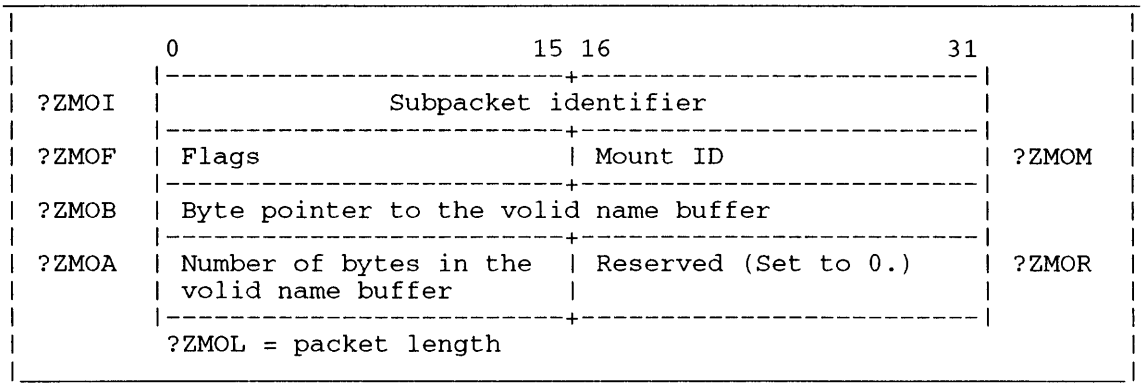


Figure 2–151. Structure of Mounted Command Subpacket

Table 2–130. Contents of Mounted Command Subpacket

Offset	Contents
?ZMOI (double-word)	Subpacket identifier. Place ?ZMOZ here.
?ZMOF	Flags word. It contains the bit flags that indicate default choices to the operating system. ?ZY10 = 1B(?ZZ10) -- Default mount ID.
?ZMOM	Specify the mount ID.
?ZMOB (double-word)	Supply a byte pointer to the volid name buffer.
?ZMOA	Specify the number of bytes in the volid name buffer.
?ZMOR	Reserved. (Set to 0.)

Offset ?ZXNA of the main ?OPEX packet in Figure 2–124 must contain the byte address of the queue name buffer for the mountstatus function. To obtain all the mount queue names, use the ?XFNQN (get queue names) function of system call ?EXEC.

Figure 2–152 shows the structure of the mountstatus command subpacket, and Table 2–131 describes its contents.

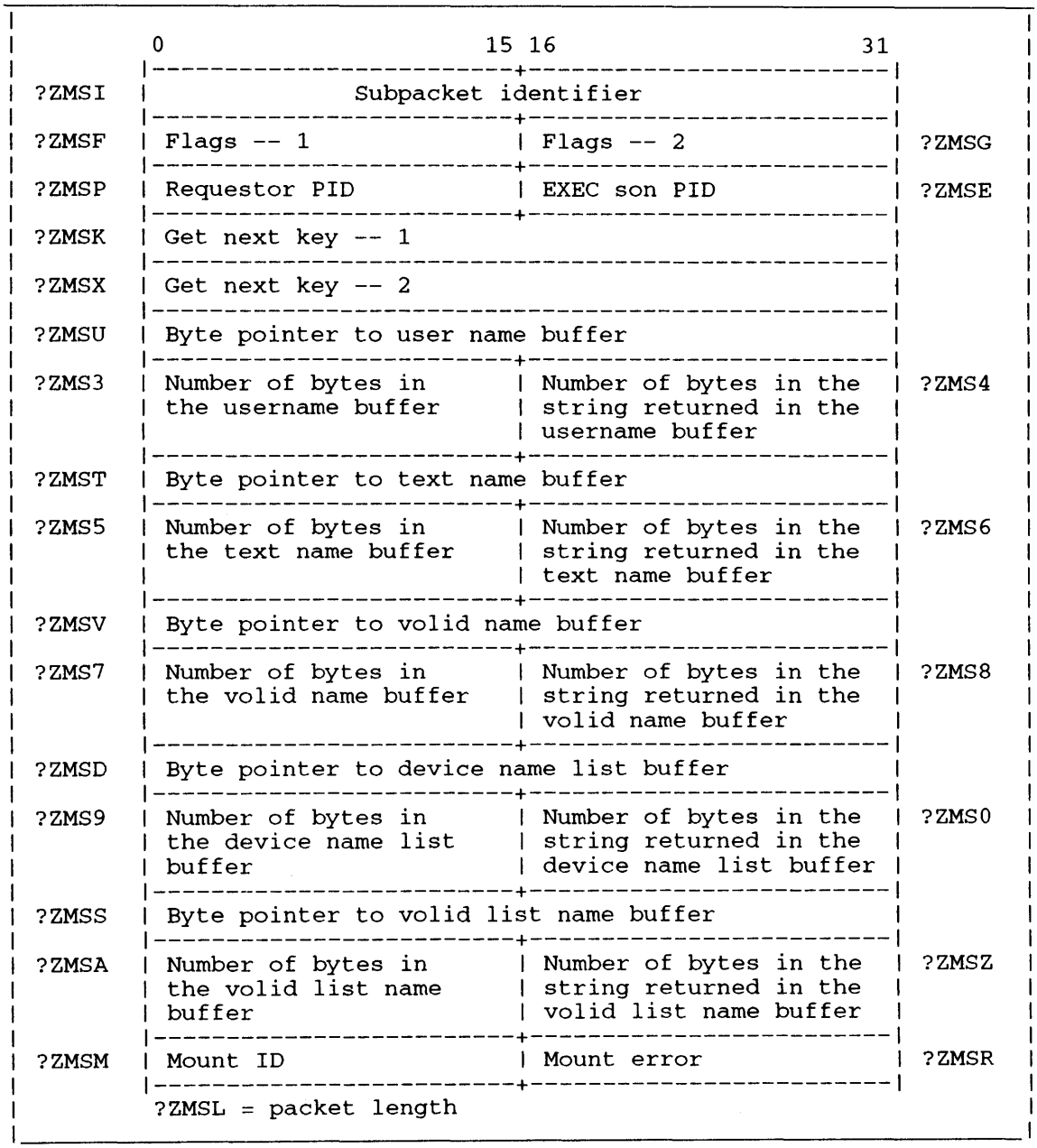


Figure 2–152. Structure of Mountstatus Command Subpacket

Table 2–131. Contents of Mountstatus Command Subpacket

Offset	Contents
?ZMSI (double-word)	Subpacket identifier. Place ?ZMSQ here.
?ZMSF	Flags word. It contains bit flags that indicate default choices to the operating system. You must set either bit ?ZY20 or bit ?ZY21. ?ZY20 = 1B(?ZZ20) -- You want the status of all requests, both inactive and active. ?ZY21 = 1B(?ZZ21) -- You want the status by mount ID. ?ZJ22 = 0B(?ZZ22) -- An output bit; explicit labeled mount request. ?ZY22 = 1B(?ZZ22) -- An output bit; implicit labeled mount request. ?ZY23 = 1B(?ZZ23) -- An output bit; unit mount request. ?ZY24 = 1B(?ZZ24) -- An output bit; waiting to be dismounted. ?ZY25 = 1B(?ZZ25) -- An output bit; next volume. ?ZY26 = 1B(?ZZ26) -- An output bit; mount error. ?ZY27 = 1B(?ZZ27) -- An output bit; specific volume. ?ZY28 = 1B(?ZZ28) -- An output bit; extend volid list. ?ZY29 = 1B(?ZZ29) -- An output bit; read only.
?ZMSG	Flags word. It contains bit flags that the operating system sets. ?ZY2G = 1B(?ZZ2G) -- An output bit; density is 800 bpi. ?ZY2H = 1B(?ZZ2H) -- An output bit; density is 1600 bpi. ?ZY2I = 1B(?ZZ2I) -- An output bit; density is 6250 bpi. ?ZY2J = 1B(?ZZ2J) -- An output bit; automatic density matching.
?ZMSP	The operating system returns the PID of the requestor.
?ZMSE	The operating system returns the PID of the immediate son of EXEC.
?ZMSK (double-word)	Get next entry key. The operating system uses this offset to hold the key indicator for EXEC.
?ZMSX (double-word)	Second get next entry key. The operating system uses this offset to hold the second key indicator for EXEC.

(continued)

Table 2–131. Contents of Mountstatus Command Subpacket

Offset	Contents
?ZMSU (double-word)	Supply a byte pointer to the buffer that receives the username.
?ZMS3	Specify a number of bytes to accommodate the largest possible username.
?ZMS4	The operating system returns the number of bytes in the username that it placed in the username buffer.
?ZMST (double-word)	Supply a byte pointer to the buffer that receives the request text.
?ZMS5	Specify a number of bytes to accommodate the largest possible request text.
?ZMS6	The operating system returns the number of bytes in the text that it placed in the request text buffer.
?ZMSV (double-word)	Supply a byte pointer to the buffer that receives the volid.
?ZMS7	Specify a number of bytes to accommodate the largest possible volid.
?ZMS8	The operating system returns the number of bytes in the volid that it placed in the volid buffer.
?ZMSD (double-word)	Supply a byte pointer to the buffer that receives the device name. The content is a list of device names that are separated by nulls; a pair of nulls terminates the list.
?ZMS9	Specify a number of bytes to accommodate the largest possible device name. The current maximum size of a device name list is 1024. bytes.
?ZMS0	The operating system returns the number of bytes in the data that it placed in the device list buffer.
?ZMSS (double-word)	Supply a byte pointer to the buffer that receives the volid list.
?ZMSA	Specify a number of bytes to accommodate the largest possible volid list.
?ZMSZ	The operating system returns the number of bytes in the volid list that it placed in the volid list buffer.
?ZMSM	Supply the mount ID.
?ZMSR	The operating system returns the mount error code.

(concluded)

Open Command

This function does not require a subpacket. Instead, supply a byte pointer to the queue name buffer that contains the queue name for the open function. This byte pointer belongs in offset ?ZXNA of the ?OPEX main packet in Figure 2–124.

Operator Command

Offset ?ZXNA of the main ?OPEX packet in Figure 2–124 must contain the byte address of the device name buffer for the operator function.

Figure 2–153 shows the structure of the operator command subpacket, and Table 2–132 describes its contents.

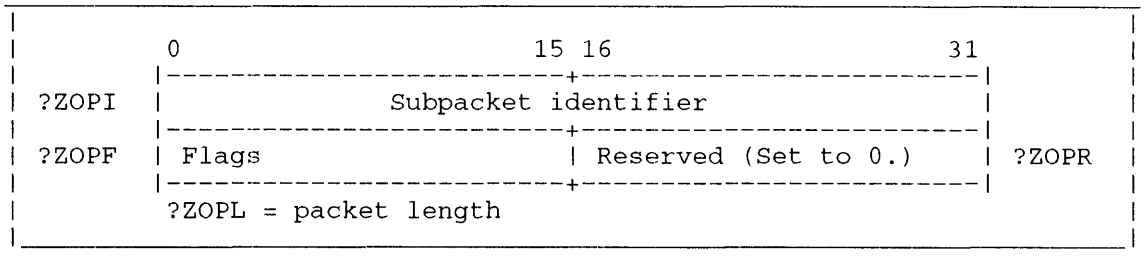


Figure 2–153. Structure of Operator Command Subpacket

Table 2–132. Contents of Operator Command Subpacket

Offset	Contents
?ZOPI (double-word)	Subpacket identifier. Place ?ZOPZ here.
?ZOPF	Flags word. It contains bit flags with additional information to the operating system and EXEC for function ?ZOP. ?ZJ30 = 0B(?ZZ30) -- There is no operator. ?ZY30 = 1B(?ZZ30) -- There is an operator on duty.
?ZOPR	Reserved. (Set to 0.)

Offset ?ZXNA of the main ?OPEX packet in Figure 2–124 must contain the byte address of the device name buffer.

If you want to pause all default batch input streams, follow these steps:

1. Set Bit ?ZZQO in offset ?ZPAF on.
2. Set the remaining bits in offset ?ZPAF off.
3. Place zero in offset ?ZPAS.
4. In the main packet, set the following to zero:
   - Offset ?ZXNA
   - Offset ?ZXNB
   - Bit ?ZZA0 in offset ?ZXFG

Suppose you want to pause all default streams associated with a batch queue you created via the ?ZCR function of ?OPEX. You have also specified the name of the queue and type=batch. Then, place zero in offsets ?ZPAF and ?ZPAS. In the main packet the offsets ?ZXNA, ?ZXNB, and ?ZXFG do not contain zero.

Figure 2–154 shows the structure of the pause command subpacket, and Table 2–133 describes its contents.

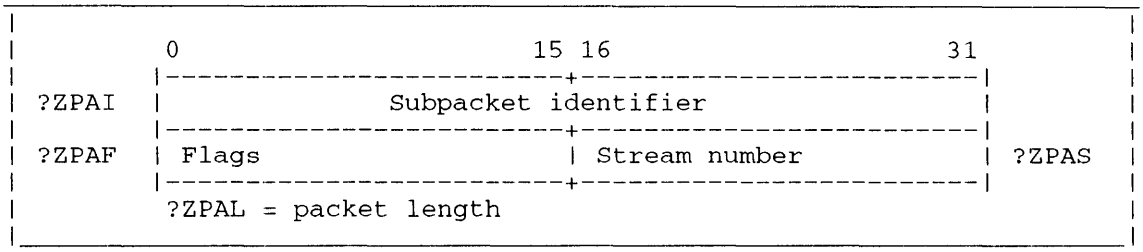


Figure 2–154. Structure of Pause Command Subpacket

Table 2–133. Contents of Pause Command Subpacket

Offset	Contents
?ZPAI (double-word)	Subpacket identifier. Place ?ZPAZ here.
?ZPAF	Flags word. It contains bit flags that indicate default choices to the operating system. ?ZYQ0 = 1B(?ZZQ0) -- Pause all default batch input streams.
?ZPAS	Supply the stream number. The default (the value 0) is to pause all batch streams.

Premount Command

Offset ?ZXNA of the main ?OPEX packet in Figure 2–124 must contain the byte address of the device name buffer for the premount function.

Figure 2–155 shows the structure of the premount command subpacket, and Table 2–134 describes its contents.

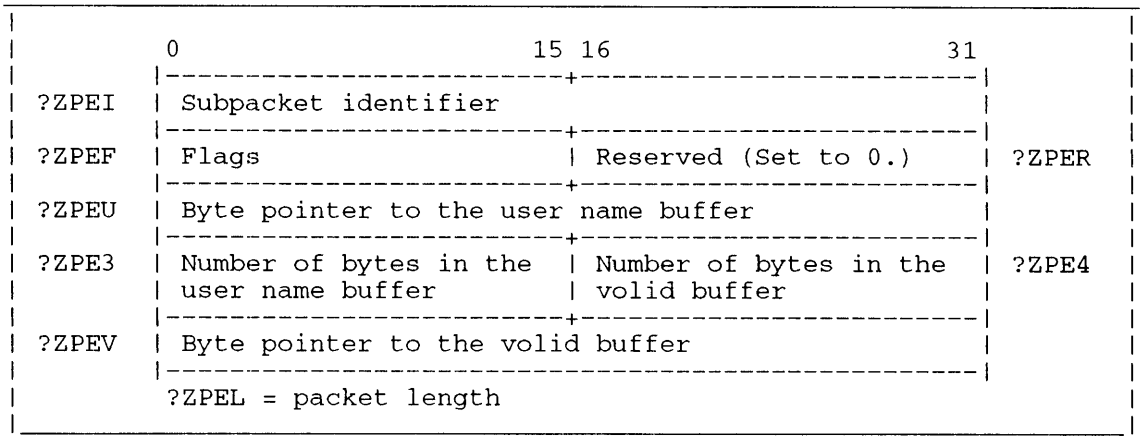


Figure 2–155. Structure of Premount Command Subpacket

Table 2–134. Contents of Premount Command Subpacket

Offset	Contents
?ZPEI (double-word)	Subpacket identifier. Place ?ZPEZ here.
?ZPEF	Flags word. It contains bit flags with additional information to the operating system and EXEC for function ?ZPE. ?ZY40 = 1B(?ZZ40) -- IBM switch.
?ZPER	Reserved (Set to 0.)
?ZPEU (double-word)	Supply a byte pointer to the buffer that receives the username.
?ZPE3	Specify the number of bytes in the username buffer.
?ZPE4	Specify the number of bytes in the volid buffer.
?ZPEV (double-word)	Supply a byte pointer to the buffer that receives the volid.

Priority Command

Offset ?ZXNA of the main ?OPEX packet in Figure 2–124 must contain the byte address of the device/queue or cooperative simple process name buffer for the priority function.

There are two correspondences between the CLI syntax for the priority command and the ?OPEX equivalent. They are next.

```
CONTROL @EXEC PRIORITY[/SWITCH] <QUEUE NAME>  [STREAM #]  [PRIORITY]
?OPEX         FUNC    (OPEX_PACKET.NAME)      ARGUMENT_1  ARGUMENT_2

CONTROL @EXEC PRIORITY[/SWITCH] @<DEVICENAME> [PRIORITY]
?OPEX         FUNC    (OPEX_PACKET.NAME)      ARGUMENT_2
```

Figure 2–156 shows the structure of the priority command subpacket, and Table 2–135 describes its contents.

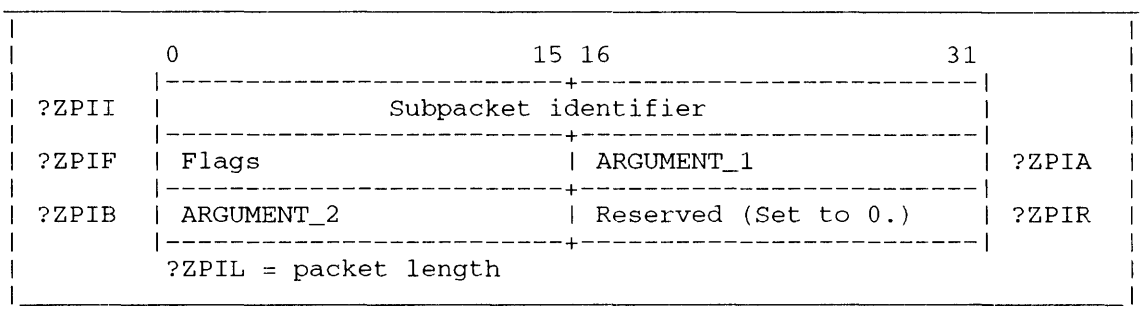


Figure 2–156. Structure of Priority Command Subpacket

Table 2–135. Contents of Priority Command Subpacket

Offset	Contents
?ZPII (double-word)	Subpacket identifier. Place ?ZPIZ here.
?ZPIF	Flags word. It contains the bit flags with additional information to the operating system and EXEC for function ?ZPI. ?ZYR0 = 1B(?ZZR0) -- Process type is swappable. ?ZYR1 = 1B(?ZZR1) -- Process type is pre-emptible. ?ZYR2 = 1B(?ZZR2) -- Process type is resident.
?ZPIA	ARGUMENT_1. If you specify a queue name, supply a stream number here. If you specify a device name, then the operating system ignores this offset. (Set it to 0.)
?ZPIB	ARGUMENT_2. Supply the priority value.
?ZPIR	Reserved. (Set to 0.)

Prompts Command

Bit ?ZZA0 of offset ?ZXFG of the main ?OPEX packet in Figure 2–124 must not be set, since the prompts function does not require a buffer to pass information. Consequently, the operating system ignores offset ?ZXNA of the main ?OPEX packet.

Figure 2–157 shows the structure of the prompts command subpacket, and Table 2–136 describes its contents.

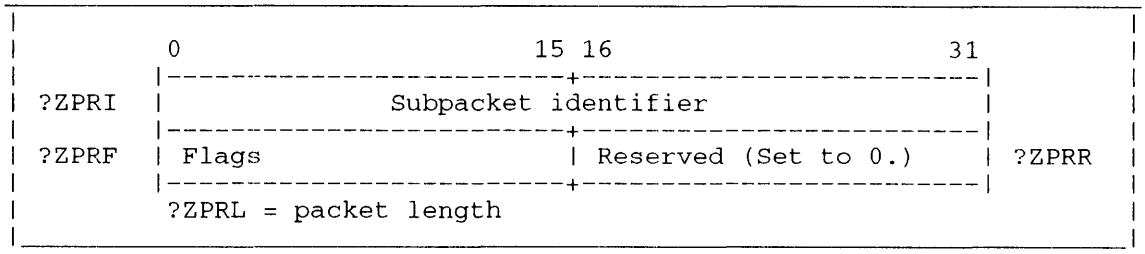


Figure 2–157. Structure of Prompts Command Subpacket

Table 2–136. Contents of Prompts Command Subpacket

Offset	Contents
?ZPRI (double-word)	Subpacket identifier. Place ?ZPRZ here.
?ZPRF	Flags word. It contains the bit flags with additional information to the operating system and EXEC for function ?ZPR. ?ZJ90 = 0B(?ZZ90) -- Prompts off. ?ZY90 = 1B(?ZZ90) -- Prompts on.
?ZPRR	Reserved. (Set to 0.)

Purge Command

This function does not require a subpacket. Instead, supply a byte pointer to the queue name buffer. This buffer contains the queue name for the purge function. Supply the byte pointer in offset ?ZXNA of the ?OPEX main packet in Figure 2–124.

Qpriority Command

Offset ?ZXNA of the main ?OPEX packet in Figure 2–124 must contain the byte address of the device/queue name buffer for the qpriority function.

There are two correspondences between the CLI syntax for the qpriority command and the ?OPEX equivalent. They are next.

```
CONTROL @EXEC QPRIORITY [QUEUE NAME]        [STREAM #] [HI PRIO] [LO PRIO]
?OPEX         FUNC      (OPEX_PACKET.NAME) ARG_1      ARG_2     ARG_3

CONTROL @EXEC QPRIORITY @DEVICENAME         [HIGH PRIORITY] [LOW PRIORITY]
?OPEX         FUNC      (OPEX_PACKET.NAME) ARG_2           ARG_3
```

Figure 2–158 shows the structure of the qpriority command subpacket, and Table 2–137 describes its contents.

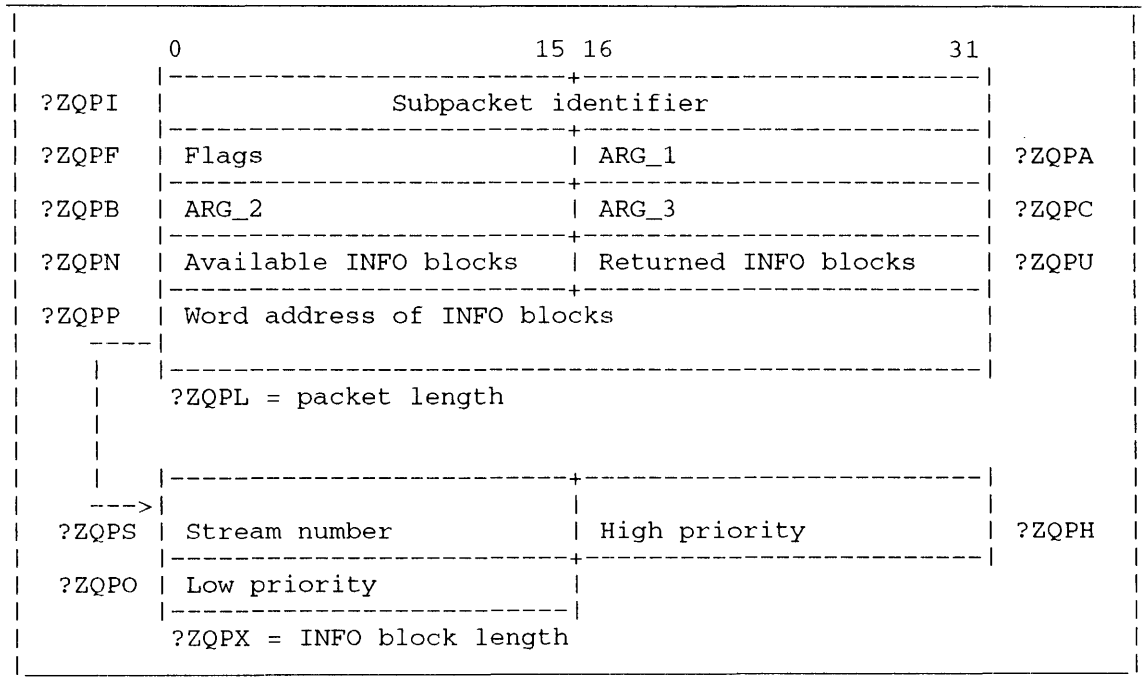


Figure 2–158. Structure of Qpriority Command Subpacket

Table 2–137. Contents of Qpriority Command Subpacket

Offset	Contents
?ZQPI (double-word)	Subpacket identifier. Place ?ZQPZ here.
?ZQPF	Flags word. It contains bit flags that indicate default choices to the operating system. ?ZYS0 = 1B(?ZZS0) -- All default batch input streams. ?ZJS1 = 0B(?ZZS1) -- Display information. ?ZYS1 = 1B(?ZZS1) -- Set information.
?ZQPA	ARG_1. If you specify a queue name, supply a stream number. If you want to display information for all streams associated with a queue, supply 0. If you specify a device name, then the operating system ignores this offset. (Set it to 0.)
?ZQPB	ARG_2. Supply the high-priority value. If you specify ?ZJS1 in offset ?ZQPF, supply zero.
?ZQPC	ARG_3. Supply the low-priority value. If you specify ?ZJS1 in offset ?ZQPF, supply zero.
?ZQPN	Specify the number of available INFO blocks, but only if offset ?ZQPF contains ?ZJS1 (to display information). Otherwise, supply 0.
?ZQPU	The operating system returns the actual number of INFO blocks, but only if offset ?ZQPF contains ?ZJS1 (to display information. Otherwise, supply 0.
?ZQPP	Supply the word address of the first of the INFO blocks, but only if offset ?ZQPF contains ?ZJS1 (to display information). Otherwise, supply 0.
?ZQPS	The operating system returns the INFO block stream number.
?ZQPH	The operating system returns the INFO block high-priority value.
?ZQPO	The operating system returns the INFO block low-priority value.

Refused Command

Bit ?ZZA0 of offset ?ZXFG of the main ?OPEX packet in Figure 2–124 must not be set, since the refused function does not require a buffer to pass information. Consequently, the operating system ignores offset ?ZXNA of the main ?OPEX packet.

Figure 2–159 shows the structure of the refused command subpacket, and Table 2–138 describes its contents.

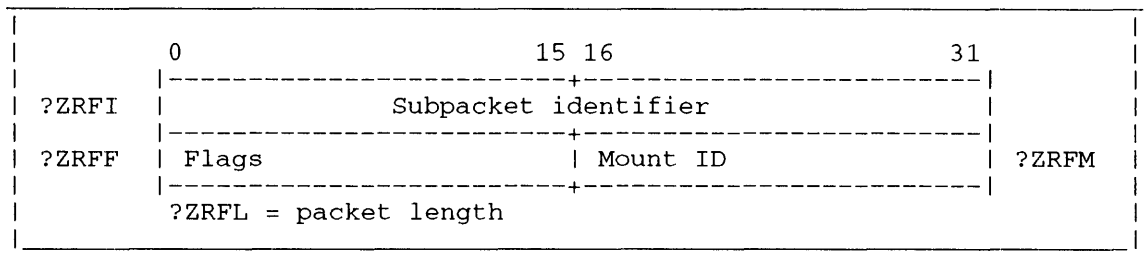


Figure 2–159. Structure of Refused Command Subpacket

Table 2–138. Contents of Refused Command Subpacket

Offset	Contents
?ZRFI (double-word)	Subpacket identifier. Place ?ZRFZ here.
?ZRFF	Flags word. It contains bit flags that indicate default choices to the operating system. ?ZY50 = 1B(?ZZ50) -- Default mount request.
?ZRFM	Supply the mount ID. If you specify the default in offset ?ZRFF, then it applies to the default mount request.

Release Command

Offset ?ZXNA of the main ?OPEX packet in Figure 2–124 must contain the byte address of the device name buffer for the release function.

Figure 2–160 shows the structure of the release command subpacket, and Table 2–139 describes its contents.

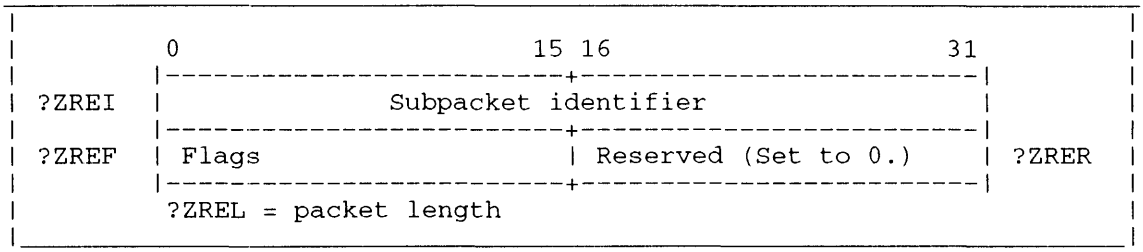


Figure 2–160. Structure of Release Command Subpacket

Table 2–139. Contents of Release Command Subpacket

Offset	Contents
?ZREI (double-word)	Subpacket identifier. Place ?ZREZ here.
?ZREF	Flags word. It contains bit flags with additional information to the operating system and EXEC for function ?ZRE. ?ZY60 = 1B(?ZZ60) -- All switch.
?ZRER	Reserved. (Set to 0.)

Restart Command

Offset ?ZXNA of the main ?OPEX packet in Figure 2–124 must contain the byte address of the device/queue name buffer for the restart function.

There are two correspondences between the CLI syntax for the restart command and the ?OPEX equivalent. They are next.

```
CONTROL @EXEC RESTART [QUEUE NAME]          [STREAM #]
?OPEX         FUNC    (OPEX_PACKET.NAME)    ARGUMENT_1

CONTROL @EXEC RESTART @DEVICENAME           [BEG. PAGE #]   [END PAGE #]
?OPEX         FUNC    (OPEX_PACKET.NAME)    ARGUMENT_1      [ARGUMENT_2]
```

Figure 2–161 shows the structure of the restart command subpacket, and Table 2–140 describes its contents.

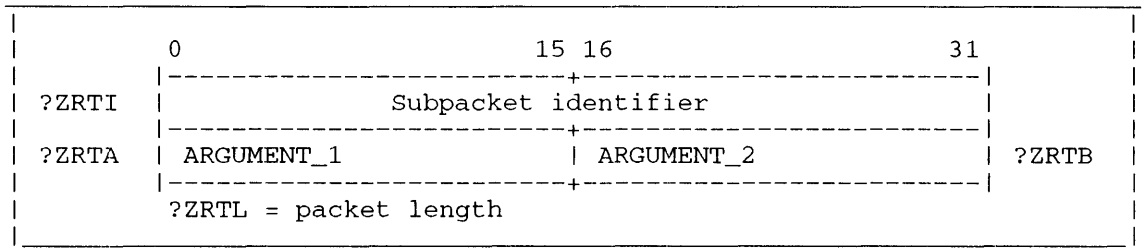


Figure 2–161. Structure of Restart Command Subpacket

Table 2–140. Contents of Restart Command Subpacket

Offset	Contents
?ZRTI (double-word)	Subpacket identifier. Place ?ZRTZ here.
?ZRTA	ARGUMENT_1. If you specify a queue name, supply a stream number here. If you specify a device name, then supply the beginning page number. The default is the first page; to specify this, supply 0.
?ZRTB	ARGUMENT_2. If you specify a queue name, the operating system ignores this offset. (Set it to 0.) If you specify a device name, then supply the ending page number. The default is the last page; to specify this, supply 0.

Silence Command

Offset ?ZXNA of the main ?OPEX packet in Figure 2–124 must contain the byte address of the device/queue name buffer for the silence function.

Figure 2–162 shows the structure of the silence command subpacket, and Table 2–141 describes its contents.

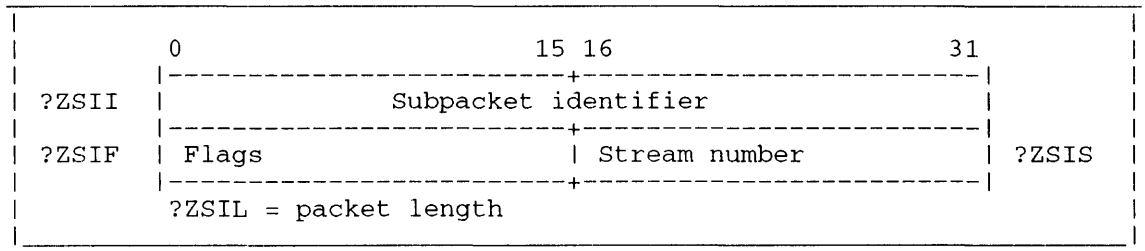


Figure 2–162. Structure of Silence Command Subpacket

Table 2–141. Contents of Silence Command Subpacket

Offset	Contents
?ZSII (double-word)	Subpacket identifier. Place ?ZSIZ here.
?ZSIF	Flags word. It contains bit flags that indicate default choices to the operating system. ?ZYT0 = 1B(?ZZT0) -- Silence all batch input streams.
?ZSIS	Supply the number of the stream if bit ?ZZT0 in offset ?ZSIF is off. Otherwise, supply 0.

Spoolstatus Command

Offset ?ZXNA of the main ?OPEX packet in Figure 2–124 must contain the byte address of the device/queue name buffer for the spoolstatus function. To obtain the status of all spooled devices, set bit ?ZZK0 of offset ?ZSPF of the subpacket. If a queue name is specified in the main ?OPEX packet, a device name list will be returned in the subpacket.

Figure 2–163 shows the structure of the spoolstatus command subpacket, and Table 2–142 describes its contents.

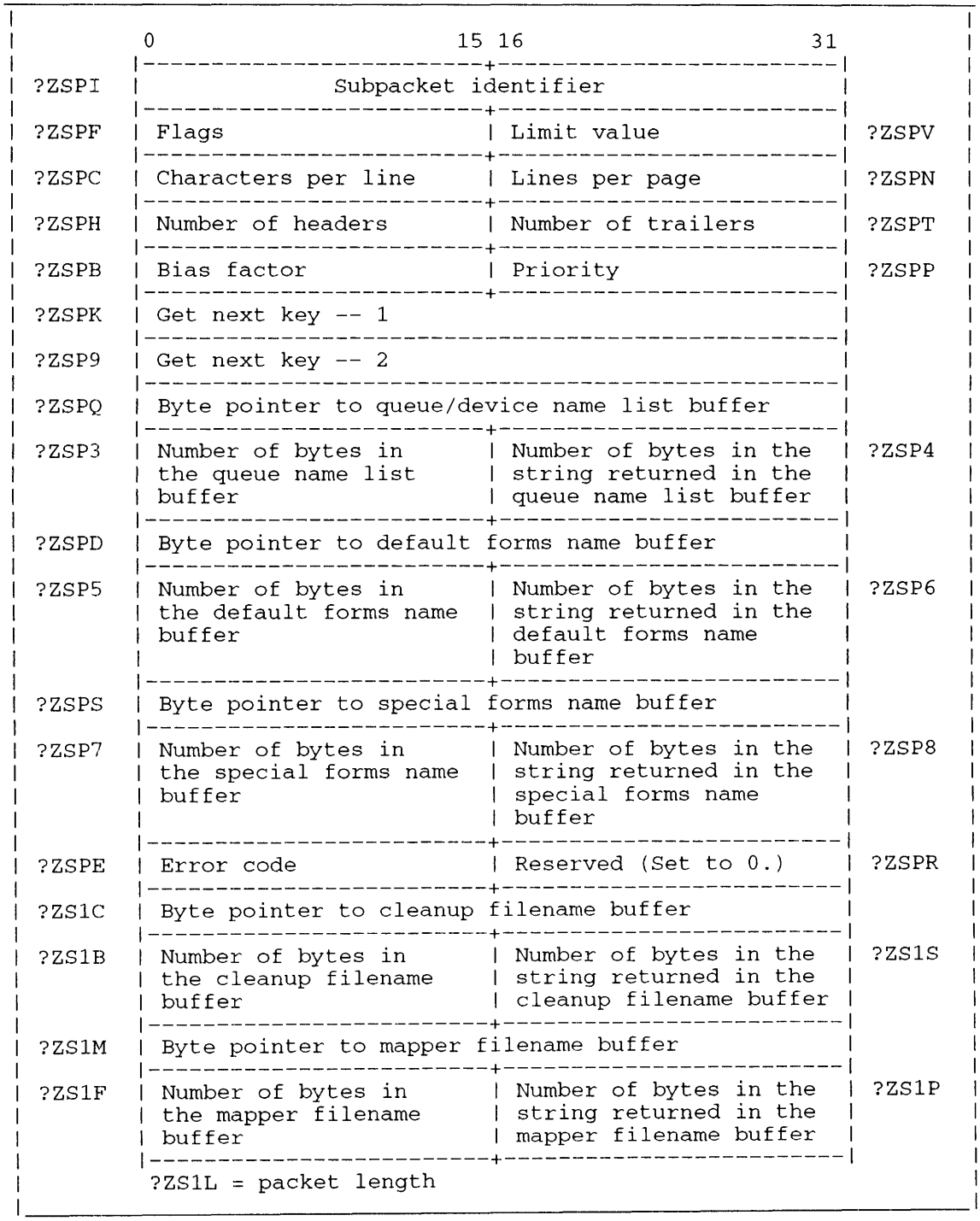


Figure 2–163. Structure of Spoolstatus Command Subpacket

Table 2–142. Contents of Spoolstatus Command Subpacket

Offset	Contents
?ZSPI (double-word)	Subpacket identifier. Place ?ZSPZ here.
?ZSPF	Flags word. It contains bit flags that indicate default choices to the operating system. ?ZYK0 = 1B(?ZZK0) -- You want the spool status of all devices. ?ZYK1 = 1B(?ZZK1) -- An output bit; even mode is enabled. ?ZYK2 = 1B(?ZZK2) -- An output bit; binary mode is enabled. ?ZYK3 = 1B(?ZZK3) -- An output bit; uppercase is enabled. ?ZYK4 = 1B(?ZZK4) -- An output bit; new line conversion is enabled. ?ZYK5 = 1B(?ZZK5) -- An output bit; process type is swappable. ?ZYK6 = 1B(?ZZK6) -- An output bit; process type is pre-emptible. ?ZYK7 = 1B(?ZZK7) -- An output bit; process type is resident. ?ZYK8 = 1B(?ZZK8) -- An output bit; default forms are specified. ?ZYK9 = 1B(?ZZK9) -- An output bit; special forms are specified. ?ZYKA = 1B(?ZZKA) -- An output bit; limiting is enabled (the default value). ?ZYKB = 1B(?ZZKB) -- An output bit; print or plot device. ?ZYKC = 1B(?ZZKC) -- An output bit; 8-bit mode is enabled.
?ZSPV	The operating system returns the limit value.
?ZSPC	The operating system returns the number of characters per line.
?ZSPN	The operating system returns the number of lines per page.
?ZSPH	The operating system returns the number of headers.
?ZSPT	The operating system returns the number of trailers.
?ZSPB	The operating system returns the bias factor.
?ZSPP	The operating system returns the priority.

(continued)

Table 2–142. Contents of Spoolstatus Command Subpacket

Offset	Contents
?ZSPK (double-word)	First get next key entry. Set to 0 for the first call, used by the operating system thereafter for the next key entry for EXEC.
?ZSP9 (double-word)	Second get next key entry. Set to 0 for the first call, used by the operating system thereafter for the next key entry for EXEC.
?ZSPQ (double-word)	The operating system returns a byte pointer to the queue name buffer. In this buffer, nulls separate the queue names and a pair of nulls terminates the queue names.
?ZSP3	Specify the number of bytes in the queue name buffer. The largest possible queue name list is 1024. bytes.
?ZSP4	The operating system returns the number of bytes in the queue name list that it placed in the queue name buffer.
?ZSPD (double-word)	Supply a byte pointer to the buffer that receives the default forms name.
?ZSP5	Specify a number of bytes to accommodate the largest possible default forms name.
?ZSP6	The operating system returns the number of bytes in the default forms filename that it placed in the default forms name buffer.
?ZSPS (double-word)	Supply a byte pointer to the buffer that receives the special forms name.
?ZSP7	Specify a number of bytes to accommodate the largest possible special forms name.
?ZSP8	The operating system returns the number of bytes in the special forms filename that it placed in the special forms name buffer.

(continued)

Table 2–142. Contents of Spoolstatus Command Subpacket

Offset	Contents
?ZSPE	The operating system returns an error code that EXEC gave to it.
?ZSPR	Reserved. (Set to 0.)
?ZS1C (double-word)	Supply a byte pointer to the buffer that receives the cleanup filename.
?ZS1B	Specify a number of bytes to accommodate the largest possible cleanup filename.
?ZS1S	The operating system returns the number of bytes in the cleanup filename that it placed in the cleanup filename buffer.
?ZS1M (double-word)	Supply a byte pointer to the buffer that receives the mapper filename.
?ZS1F	Specify a number of bytes to accommodate the largest possible mapper filename.
?ZS1P	The operating system returns the number of bytes in the mapper filename that it placed in the mapper filename buffer.

(concluded)

Stack Command

Offset ?ZXNA of the main ?OPEX packet in Figure 2–124 must contain the byte address of a pathname buffer. This buffer contains the pathname for the stack function.

Figure 2–164 shows the structure of the stack command subpacket, and Table 2–143 describes its contents.

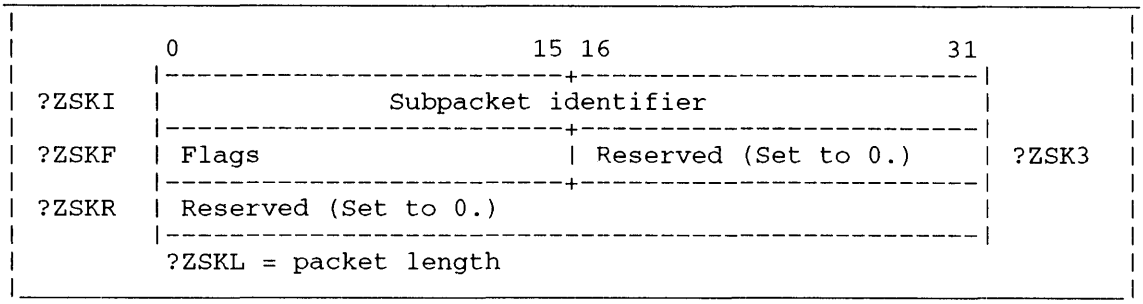


Figure 2–164. Structure of Stack Command Subpacket

Table 2–143. Contents of Stack Command Subpacket

Offset	Contents
?ZSKI (double-word)	Subpacket identifier. Place ?ZSKZ here.
?ZSKF	Flags word. It contains the bit flags with additional information to the operating system and EXEC for function ?ZSK.
	Reserved. (Set to 0.)
?ZSK3	Reserved. (Set to 0.)
?ZSKR (double-word)	Reserved. (Set to 0.)

Start Command

The EXEC START (?OPEX ?ZSR) command associates a queue with a device and a cooperative. Offset ?ZXNA of the main ?OPEX packet in Figure 2–124 must contain the byte pointer of the device name.

Figure 2–165 shows the structure of the start command subpacket, and Table 2–144 describes its contents.

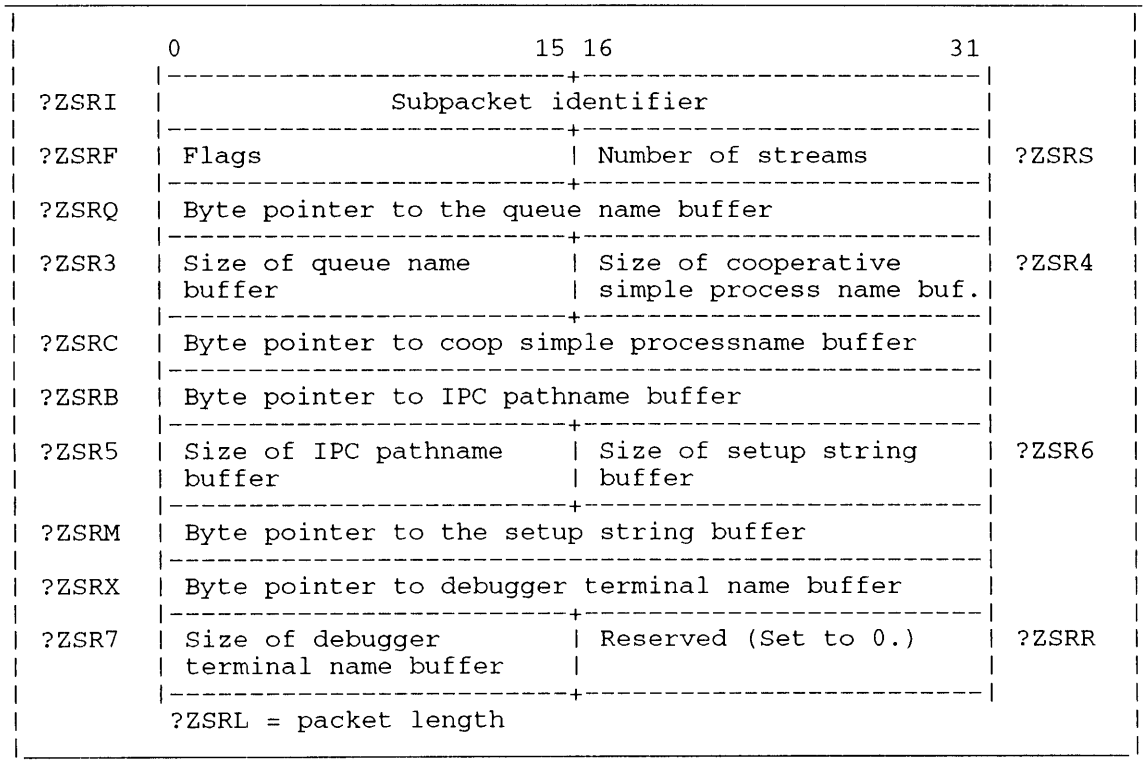


Figure 2–165. Structure of Start Command Subpacket

When EXEC is to communicate via an IPC file, set bit ?ZZU7 of the ?ZSRF flag word. When EXEC is to create (?PROC) a cooperative process or communicate with an existing cooperative that is unknown to EXEC, set bit ?ZZU2 of the ?ZSRF flag word.

Load offset ?ZSRC (or ?ZSRB) with the byte pointer to the simple processname (or IPC pathname) buffer. Finally, load offset ?ZSR4 (or ?ZSR5) with the number of bytes in the buffer.

The meaning of ?ZSRM and ?ZSR6 have been generalized. ?ZSRM now contains the byte address of a setup string which is passed to the cooperative in the START IPC. ?ZSR6 contains the byte length (max. 32) of this setup string. This string maintains its current use of containing the mapper file name when a printer is started.

Table 2–144. Contents of Start Command Subpacket

Offset	Contents
?ZSRI (double-word)	Subpacket identifier. Place ?ZSRZ here.
?ZSRF	Flags word. It contains bit flags with additional information to the OS and EXEC for function ?ZSR. ?ZYU0 = 1B(?ZZU0) -- New line switch. ?ZYU1 = 1B(?ZZU1) -- 8-bit switch. ?ZYU2 = 1B(?ZZU2) -- Process switch. ?ZYU3 = 1B(?ZZU3) -- Streams switch. ?ZYU4 = 1B(?ZZU4) -- Name switch. ?ZYU5 = 1B(?ZZU5) -- Upper argument specified. ?ZYU6 = 1B(?ZZU6) -- Reserved (Set to 0.) ?ZYU7 = 1B(?ZZU7) -- IPC switch. ?ZYU8 = 1B(?ZZU8) -- 7-bit switch.
?ZSRS	Specify the number of streams.
?ZSRQ (double-word)	Supply a byte pointer to the queue name buffer.
?ZSR3	Supply the number of bytes in the queue name buffer.
?ZSR4	Supply the number of bytes in the cooperative simple process name buffer.
?ZSRC (double-word)	Supply a byte pointer to the cooperative simple process name buffer (/NAME=).
?ZSRB (double-word)	Supply a byte pointer to the IPC filename buffer (/IPC=).
?ZSR5	Supply the number of bytes in the IPC filename buffer.
?ZSR6	Supply the number of bytes in the mapper filename buffer.
?ZSRM (double-word)	Supply a byte pointer to the mapper filename buffer.
?ZSRX (double-word)	Supply a byte pointer to the debugger terminal name buffer.
?ZSR7	Supply the number of bytes in the debugger terminal name buffer.
?ZSRR	Reserved. (Set to 0.)

Status Command

Offset ?ZXNA of the main ?OPEX packet in Figure 2–124 must contain the byte address of the device/queue name buffer for the status function. To obtain the status of all default batch streams and have the queue name returned in the main packet, set bit ?ZZV0 of offset ?ZSSF of the subpacket.

Figure 2–166 shows the structure of the status command subpacket, and Table 2–145 describes its contents.

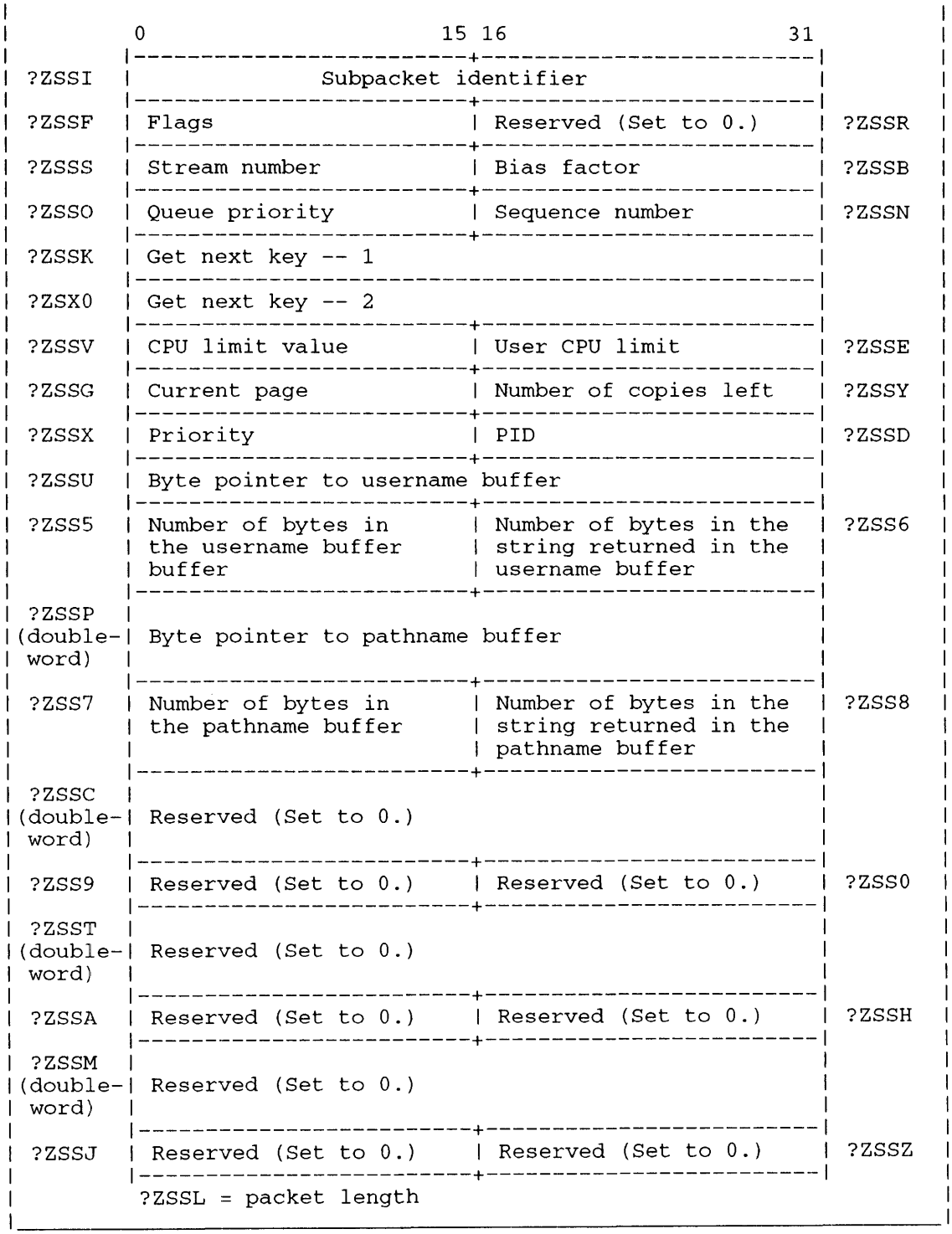


Figure 2–166. Structure of Status Command Subpacket

Table 2–145. Contents of Status Command Subpacket

Offset	Contents
?ZSSI (double-word)	Subpacket identifier. Place ?ZQSS here.
?ZSSF	Flags word. It contains bit flags that indicate default choices to the operating system. ?ZYV0 = 1B(?ZZV0) -- You want the status of all default batch input streams. ?ZJV1 = 0B(?ZZV1) -- An output bit; stream/device is idle. ?ZYV1 = 1B(?ZZV1) -- An output bit; stream/device is active. ?ZYV2 = 1B(?ZZV2) -- An output bit; process type is swappable. ?ZYV3 = 1B(?ZZV3) -- An output bit; process type is pre-emptible. ?ZYV4 = 1B(?ZZV4) -- An output bit; process type is resident. ?ZJV5 = 0B(?ZZV5) -- An output bit; 7-bit print mode. ?ZJV6 = 0B(?ZZV6) -- An output bit; stream/device is not ready. ?ZYV6 = 1B(?ZZV6) -- An output bit; stream/device is ready. ?ZYV7 = 1B(?ZZV7) -- An output bit; stream/device is paused. (If the active bit is also set, the stream/device will be paused.) ?ZYV8 = 1B(?ZZV8) -- An output bit; limiting is enabled. ?ZYV9 = 1B(?ZZV9) -- An output bit; device is being aligned. ?ZYVA = 1B(?ZZVA) -- An output bit; batch job. ?ZYVB = 1B(?ZZVB) -- An output bit; print or plot device. ?ZYVC = 1B(?ZZVC) -- An output bit; net cooperative. ?ZJVD = 0B(?ZZVD) -- An output bit; net cooperative is not ready. ?ZYVD = 1B(?ZZVD) -- An output bit; net cooperative is ready. ?ZJVE = 0B(?ZZVE) -- An output bit; net cooperative is idle. ?ZYVE = 1B(?ZZVE) -- An output bit; net cooperative is active.

(continued)

Table 2–145. Contents of Status Command Subpacket

Offset	Contents
?ZSSR	Reserved. (Set to 0.)
?ZSSS	Supply the stream number.
?ZSSB	The operating system returns the bias factor.
?ZSSO	The operating system returns the queue priority.
?ZSSN	The operating system returns the sequence number.
?ZSSK (double-word)	First get next key entry. Set to 0 for the first call, used by the operating system thereafter for the next key identifier for EXEC.
?ZSX0 (double-word)	Second get next key entry. Set to 0 for the first call, used by the operating system thereafter for the next key identifier for EXEC.
?ZSSV	The operating system returns the system's CPU limit value.
?ZSSE	The operating system returns the user-specified CPU limit value.
?ZSSG	The operating system returns the current page.
?ZSSY	The operating system returns the number of copies left.
?ZSSX	The operating system returns the priority.
?ZSSD	The operating system returns the PID.
?ZSSU (double-word)	Supply a byte pointer to the buffer to receive the username.
?ZSS5	Specify the number of bytes in the username buffer.
?ZSS6	The operating system returns the number of bytes in the string that it placed in the username buffer.
?ZSSP (double-word)	Supply a byte pointer to the buffer that receives the pathname.
?ZSS7	Specify the number of bytes in the pathname buffer.
?ZSS8	The operating system returns the number of bytes in the string that it placed in the pathname buffer.
?ZSSC (double-word)	Reserved. (Set to 0.)
?ZSS9	Reserved. (Set to 0.)

(continued)

Table 2–145. Contents of Status Command Subpacket

Offset	Contents
?ZSS0	Reserved. (Set to 0.)
?ZSST (double-word)	Reserved. (Set to 0.)
?ZSSA	Reserved. (Set to 0.)
?ZSSH	Reserved. (Set to 0.)
?ZSSM (double-word)	Reserved. (Set to 0.)
?ZSSJ	Reserved. (Set to 0.)
?ZSSZ	Reserved. (Set to 0.)

(concluded)

Stop Command

Offset ?ZXNA of the main ?OPEX packet in Figure 2–124 must contain the byte address of the device/queue name buffer for the stop function.

Figure 2–167 shows the structure of the stop command subpacket, and Table 2–146 describes its contents.

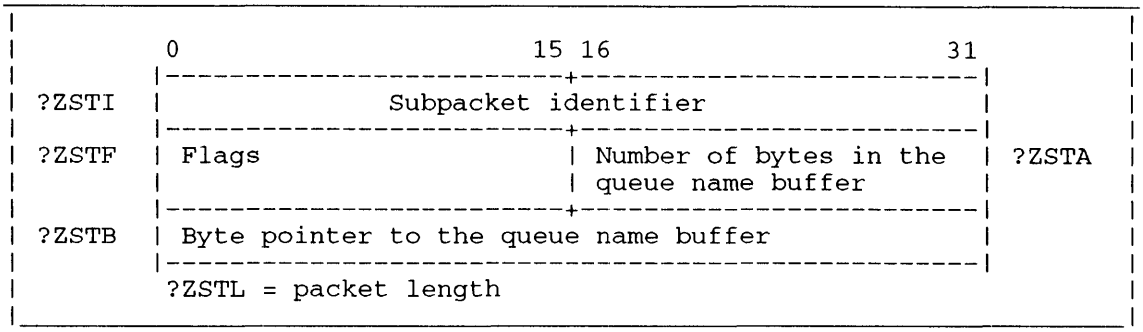


Figure 2–167. Structure of Stop Command Subpacket

Table 2–146. Contents of Stop Command Subpacket

Offset	Contents
?ZSTI (double-word)	Subpacket identifier. Place ?ZSTZ here.
?ZSTF	Flags word. It contains bit flags that indicate default choices to the operating system. ?ZYW0 = 1B(?ZZW0) -- Queue name specified.
?ZSTA	Specify the number of bytes in the queue name buffer if you need the buffer.
?ZSTB (double-word)	Supply a byte pointer to the queue name buffer if you need the buffer.

Terminate Command

This function does not require a subpacket. Instead, supply a byte pointer to the device name buffer in offset ?ZXNA of the ?OPEX main packet in Figure 2–124.

Trailers Command

Offset ?ZXNA of the main ?OPEX packet in Figure 2–124 must contain the byte address of the device name buffer for the trailers function.

Figure 2–168 shows the structure of the trailers command subpacket, and Table 2–147 describes its contents.

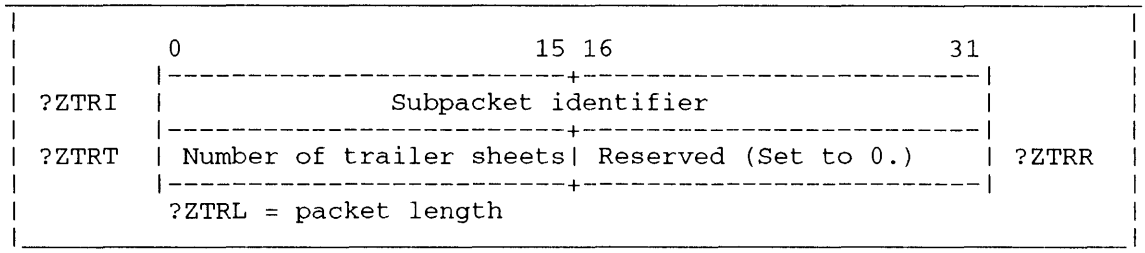


Figure 2–168. Structure of Trailers Command Subpacket

Table 2–147. Contents of Trailers Command Subpacket

Offset	Contents
?ZTRI (double-word)	Subpacket identifier. Place ?ZTRZ here.
?ZTRT	Supply the number of trailer sheets after each printed file.
?ZTRR	Reserved. (Set to 0.)

Unhold Command

Offset ?ZXNA of the main ?OPEX packet in Figure 2–124 must contain the byte address of the username buffer, if necessary, for the unhold function.

Figure 2–169 shows the structure of the unhold command subpacket, and Table 2–148 describes its contents.

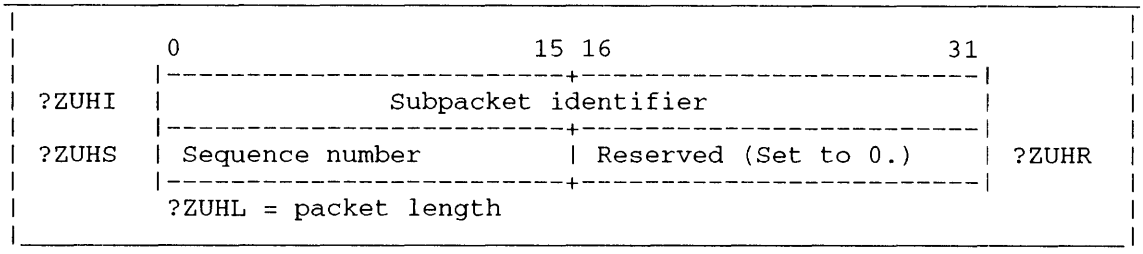


Figure 2–169. Structure of Unhold Command Subpacket

Table 2–148. Contents of Unhold Command Subpacket

Offset	Contents
?ZUHI (double-word)	Subpacket identifier. Place ?ZUHZ here.
?ZUHS	Supply the sequence number in this offset. Or, to perform the unhold function on all jobs with a given username, supply 0 here and supply a byte pointer to the username in offset ?ZXNA of the main packet of ?OPEX.
?ZUHR	Reserved. (Set to 0.)

Unitstatus Command

Offset ?ZXNA of the main ?OPEX packet in Figure 2–124 must contain the byte address of the device name buffer for the unitstatus function.

Figure 2–170 shows the structure of the unitstatus command subpacket, and Table 2–149 describes its contents.

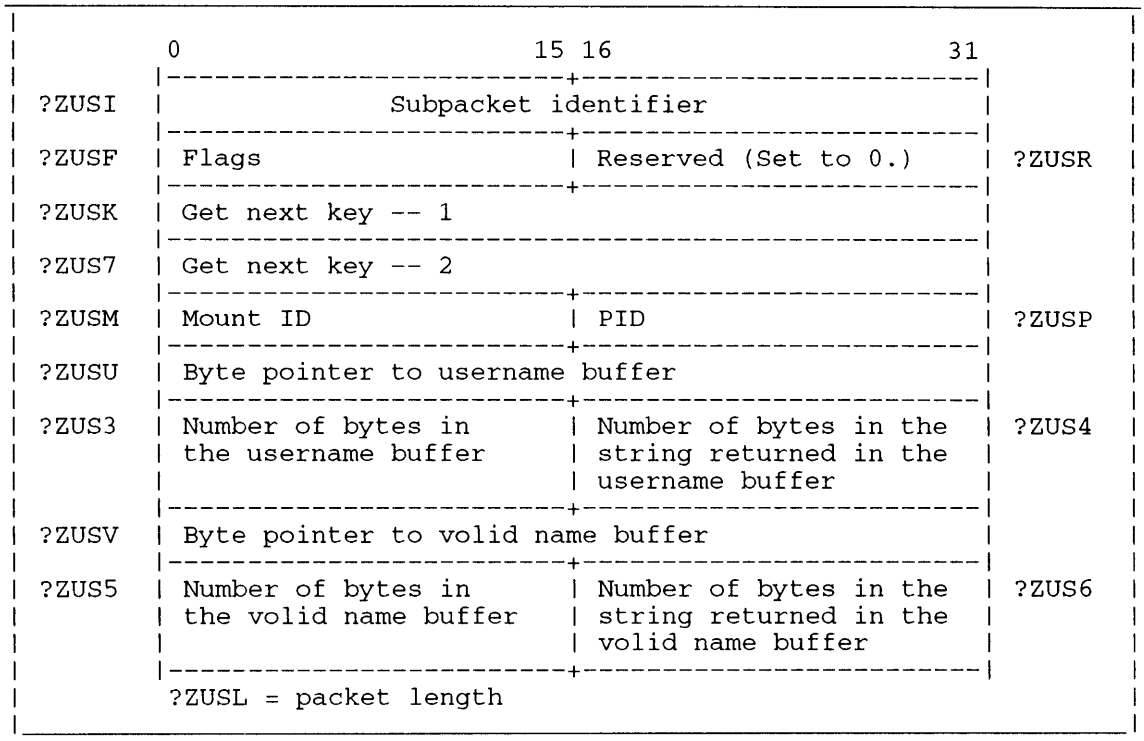


Figure 2–170. Structure of Unitstatus Command Subpacket

Table 2–149. Contents of Unitstatus Command Subpacket

Offset	Contents
?ZUSI (double-word)	Subpacket identifier. Place ?ZUSZ here.
?ZUSF	Flags word. It contains bit flags that indicate default choices to the operating system. ?ZY70 = 1B(?ZZ70) -- All units. ?ZY71 = 1B(?ZZ71) -- An output bit; unit not mounted. ?ZY72 = 1B(?ZZ72) -- An output bit; unit is premounted. ?ZY73 = 1B(?ZZ73) -- An output bit; unit is waiting to be dismounted. ?ZY74 = 1B(?ZZ74) -- An output bit; IBM format.
?ZUSR	Reserved. (Set to 0.)
?ZUSK (double-word)	Get next entry key. The operating system uses this offset to hold the key identifier for EXEC.
?ZUS7 (double-word)	Second get next entry key. The operating system uses this offset to hold the second key indicator for EXEC.
?ZUSM	The operating system returns the mount ID.
?ZUSP	The operating system returns the PID.
?ZUSU (double-word)	Supply a byte pointer to the buffer that receives the username.
?ZUS3	Specify a number of bytes to accommodate the largest possible username.
?ZUS4	The operating system returns the number of bytes in the username that it placed in the username buffer.
?ZUSV (double-word)	Supply a byte pointer to the buffer that receives the volid.
?ZUS5	Specify a number of bytes to accommodate the largest possible volid.
?ZUS6	The operating system returns the number of bytes in the volid that it placed in the volid buffer.

Unlimit Command

Offset ?ZXNA of the main ?OPEX packet in Figure 2–124 must contain the byte address of the device/queue name buffer for the unlimit function. If you take the default option and set flag bit ?ZZX0 in offset ?ZULF of the subpacket, then the unlimit function applies to all batch streams.

Figure 2–171 shows the structure of the unlimit command subpacket, and Table 2–150 describes its contents.

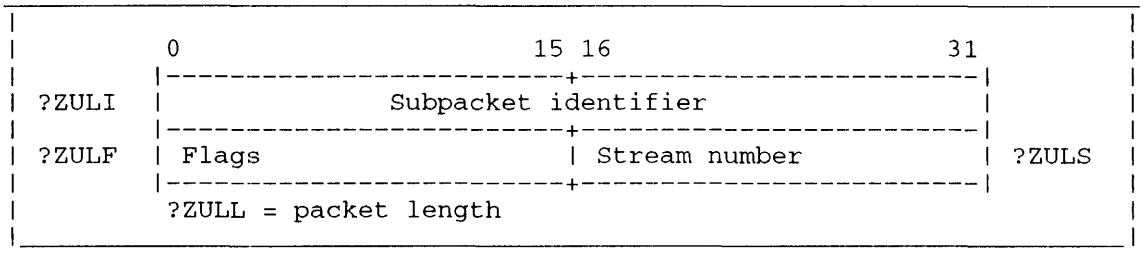


Figure 2–171. Structure of Unlimit Command Subpacket

Table 2–150. Contents of Unlimit Command Subpacket

Offset	Contents
?ZULI (double-word)	Subpacket identifier. Place ?ZULZ here.
?ZULF	Flags word. It contains bit flags that indicate default choices to the operating system. ?ZYX0 = 1B(?ZZX0) -- Unlimit all default batch input streams.
?ZULS	Supply the number of the stream if bit ?ZZX0 in offset ?ZULF is off. Otherwise, supply 0.

Unsilence Command

Offset ?ZXNA of the main ?OPEX packet in Figure 2–124 must contain the byte address of the device/queue name buffer for the unsilence function. If you take the default option and set flag bit ?ZZY0 in offset ?ZUNF of the subpacket, then the unsilence function applies to all batch streams.

Figure 2–172 shows the structure of the unsilence command subpacket, and Table 2–151 describes its contents.

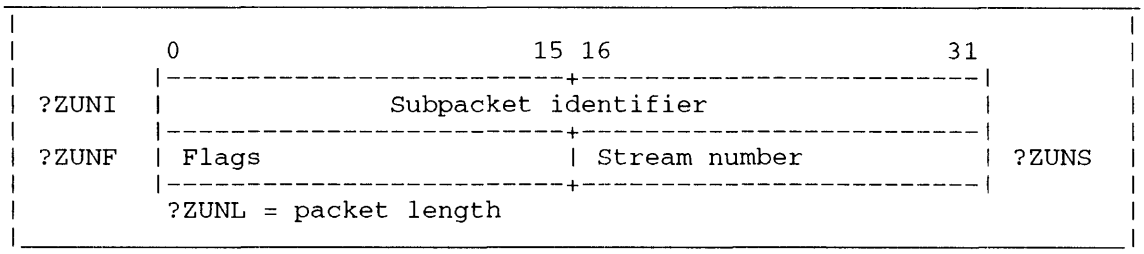


Figure 2–172. Structure of Unsilence Command Subpacket

Table 2–151. Contents of Unsilence Command Subpacket

Offset	Contents
?ZUNI (double-word)	Subpacket identifier. Place ?ZUNZ here.
?ZUNF	Flags word. It contains bit flags that indicate default choices to the operating system. ?ZYY0 = 1B(?ZZY0) -- Unsilence all default batch input streams.
?ZUNS	Supply the number of the stream if bit ?ZZY0 in offset ?ZUNF is off. Otherwise, supply 0.

User Command

The user command passes a device message between EXEC and the cooperative controlling the device referred to by ?ZXNA in the main ?OPEX packet. This allows the programmer of a cooperative to define device–oriented commands that EXEC routes to the cooperative. The action that the cooperative takes in response to the message is up to the cooperative.

Figure 2–173 shows the structure of the user–command subpacket. Table 2–152 describes its contents.

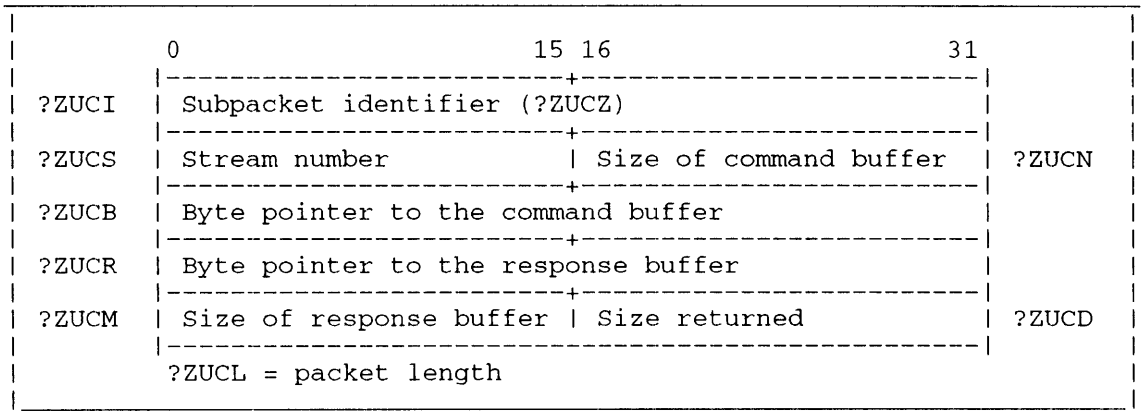


Figure 2–173. Structure of User–Command Subpacket

Table 2–152. Contents of Subpacket User–Command

Offset	Contents
?ZUCS	Set to zero or a stream number assigned to the the device.
?ZUCN	Set to the number of bytes in the command buffer. It must be less than 1024 bytes. If ?ZUCN is equal to or greater than 1024 bytes, EXEC takes the error return and sends no message to the cooperative.
?ZUCB	Specify a byte pointer to the command buffer. Bit 0 of the first word of the command buffer is reserved for use by Data General. Set the bit to zero. If you fail to do it, EXEC takes the error return and sends no message to the cooperative. Otherwise, the cooperative interface defines the form and content of the message in the command buffer. Refer to the Release Notice for more information.
?ZUCR and ?ZUCM	If you do not wish to receive a response, set ?ZUCR and ?ZUCM to 0. If you wish to receive a response, set ?ZUCR to the byte address of a response buffer and ?ZUCM to the length of the buffer. If the cooperative sends a response that is longer than the buffer, EXEC truncates the response.
?ZUCD	EXEC loads ?ZUCD with the number of bytes in the response. It will never exceed 1024 bytes.

Note

- User command does not support 16–bit programs nor is it available through the CLI (i.e., no CONTROL @EXEC USERCOMMAND).

Verbose Command

Offset ?ZXNA of the main ?OPEX packet in Figure 2–124 must contain the byte address of the device/queue name buffer for the verbose function. If you take the default option and set flag bit ?ZZZ0 in offset ?ZVEF of the subpacket, then the verbose function applies to all batch streams.

Figure 2–174 shows the structure of the verbose command subpacket, and Table 2–153 describes its contents.

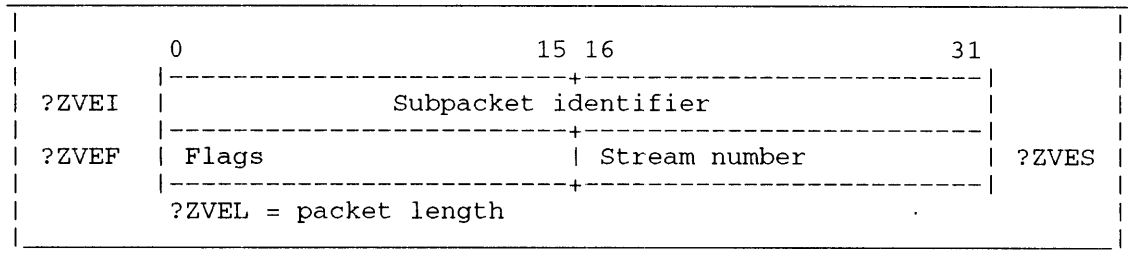


Figure 2–174. Structure of Verbose Command Subpacket

Table 2–153. Contents of Verbose Command Subpacket

Offset	Contents
?ZVEI (double-word)	Subpacket identifier. Place ?ZVEZ here.
?ZVEF	Flags word. It contains bit flags that indicate default choices to the operating system. ?ZYZ0 = 1B(?ZZZ0) -- Verbose on all default batch input streams.
?ZVES	Supply the number of the stream if bit ?ZZZ0 in offset ?ZVEF is off. Otherwise, supply 0.

Xbias Command

This function is currently undefined.

?OVEX — Releases an overlay and returns (16–bit processes only).

?OVEX
error return

Input

AC0 Contains the following:

- Bit 0 is ignored
- Overlay area number (defined in the .ENTO argument) in Bits 1 through 6
- Overlay number (defined in the .ENTO argument) in Bits 7 through 15

AC1 Reserved (Set to 0.)

AC2 Normal return address

Output

None

Error Codes in AC0

ERICM	Illegal system command
ERROO	Invalid overlay number
EROVN	Illegal overlay number

Why Use It?

?OVEX is one of the three primitive overlay system calls that release an overlay. (The others are ?OVREL and ?OVKIL.) If you load an overlay with the primitive overlay system call ?OVLOD, you must use ?OVEX, ?OVREL, or ?OVKIL to release it.

?OVEX offers two advantages over the other release system calls: you can issue ?OVEX from within an overlay (unlike ?OVREL), and you can use ?OVEX to release an overlay without killing the calling task (unlike ?OVKIL).

Who Can Use It?

There are no special process privileges needed to issue this call. You must have Read access to the program's overlay (.OL) file.

What It Does

?OVEX exits from an overlay loaded by a previous ?OVLOD system call, decrements the overlay's overlay use count (OUC), and frees the overlay area if the OUC becomes 0. ?OVEX also transfers control to an address outside the overlay. This can be the caller's return address, if the caller is returning from a subroutine within the overlay.

Note that you must specify the ?OVEX return address in AC2.

Notes

- See the descriptions of ?OVLOD, ?OVREL, ?OVKIL in this chapter.

?OVKIL — Exits from an overlay and kills the calling task (16–bit processes only).

?OVKIL
error return

Input

AC0 Contains the following:

- Bit 0 is ignored
- Overlay area number (defined in the .ENTO argument) in Bits 1 through 6
- Overlay number (defined in the .ENTO argument) in Bits 7 through 15

AC1 Reserved (Set to 0.)

AC2 Reserved (Set to 0.)

Output

None

Error Codes in AC0

ERICM	Illegal system command
ERROO	Invalid overlay number
EROVN	Illegal overlay number

Why Use It?

?OVKIL is one of the three primitive system calls for releasing an overlay. (The others are ?OVEX and ?OVREL). Use ?OVKIL if you not only want to release an overlay, but you also want to kill the task using the overlay and transfer control to the task with the next highest priority.

Who Can Use It?

There are no special process privileges needed to issue this call. You must have Read access to the program's overlay (.OL) file.

What It Does

?OVKIL decrements the target overlay's overlay use count (OUC) value, releases the overlay if the OUC becomes 0, and kills the calling task. Because ?OVKIL kills the calling task, there is no normal return for ?OVKIL.

Notes

- See the descriptions of ?OVEX and ?OVREL in this chapter.

?OVLOD

Loads and goes to an overlay (16–bit processes only).

?OVLOD
error return
normal return

Input

AC0 Contains the following:

- Bit 0 is a flag bit:

  Bit 0 = 0 on a conditional load

  Bit 0 = 1 on an unconditional load

- Overlay area number (defined in the .ENTO argument) in Bits 1 through 6
- Overlay number (defined in the .ENTO argument) in Bits 7 through 15

AC1 Contains the following:

- Bit 0 is a flag bit:

  Bit 0 = 0 to pass control to an offset within the overlay

  Bit 0 = 1 to pass control to an address within the overlay

- Target address or offset within the overlay (If Bits 1 through 15 = –1, control returns to the ?OVLOD normal return.)

AC2 Contents passed to the overlay (unless AC1 contains –1)

Output

AC0 Unchanged

AC1 Base address of the target overlay

AC2 Unchanged

Error Codes in AC0

ERADR	Illegal starting address
ERICM	Illegal system command
EROVN	Illegal overlay number

?OVLOD Continued

Why Use It?

?OVLOD is an alternate way of calling an overlay from the .OL file to an overlay area in memory. ?OVLOD and the other primitive overlay system calls give you direct control of overlay management, but they also require that you explicitly load and release each overlay. The resource system calls (?RCALL, ?KCALL, and ?RCHAIN) manage overlays automatically. Therefore, we recommend that you use resource system calls instead of primitive overlay system calls.

Who Can Use It?

There are no special process privileges needed to issue this call. You must have Read access to the program's overlay (.OL) file.

What It Does

?OVLOD loads an overlay into its overlay area and, optionally, transfers control to some point in the overlay (which you specify as an address or as an offset).

To use ?OVLOD and the other primitive overlay system calls (?OVREL, ?OVEX, and ?OVKIL), you must define each overlay with the .ENTO (overlay entry) pseudo–op.

The input status of Bit 0 in AC0 governs whether the operating system loads the overlay conditionally or unconditionally. If you specify conditional loading (Bit 0 of AC0 = 0), the operating system loads the overlay only if it is not already loaded. If the overlay is resident, a conditional ?OVLOD simply directs the operating system to increment the overlay's OUC value. If you specify unconditional loading, the operating system loads the overlay, whether or not it is resident in the overlay area.

If you set Bits 1 through 15 of AC1 to –1, ?OVLOD takes the normal return after the operating system loads the overlay. Otherwise, the operating system interprets AC1 as follows:

- If Bit 0 of AC1 = 1, then the operating system treats Bits 1 through 15 as an offset into the overlay and passes control to that offset.
- If Bit 0 of AC1 = 0, then the operating system treats Bits 1 through 15 as an absolute address in the overlay, and passes control to that address.

Note that you must set Bits 1 through 15 in AC1 to an offset if the total overlay area is a multiple of the basic overlay area. You cannot specify an absolute address in this case, because the operating system may relocate the overlay to any basic area within the total area. Thus, any absolute address you give could be invalid.

If you transfer control to the overlay (rather than the ?OVLOD normal return), the operating system passes the overlay the base address in AC1 after it executes ?OVLOD. If the overlay then issues ?OVKIL, the operating system never takes the ?OVLOD normal return, because ?OVKIL kills the calling task.

The caller receives the overlay base address (and the other output values) only if AC1 contains –1 on input or if the loaded overlay returns control to the caller with an RTN instruction. Otherwise, the output values at the ?OVLOD normal return are those passed as input values to ?OVEX.

Notes

- See the description of ?OVKIL in this chapter.

?OVREL Releases an overlay area (16–bit processes only).

?OVREL
error return
normal return

Input

AC0 Contains the following:

- Bit 0 is ignored
- Overlay area number (defined in the .ENTO argument) in Bits 1 through 6
- Overlay number (defined in the .ENTO argument) in Bits 7 through 15

AC1 Reserved (Set to 0.)

AC2 Reserved (Set to 0.)

Output

AC0 Unchanged

AC1 Undefined

AC2 Undefined

Error Codes in AC0

ERICM	Illegal system command
ERROO	Invalid overlay number
EROVN	Illegal overlay number

Why Use It?

?OVREL is one of the three primitive system calls for releasing an overlay. (The others are ?OVEX and ?OVLOD). Use ?OVREL if you want to release an overlay and maintain the calling task in the active state. (?OVKIL kills the calling task.)

Who Can Use It?

There are no special process privileges needed to issue this call. You must have Read access to the program's overlay (.OL) file.

What It Does

?OVREL decrements the overlay use count (OUC) of an overlay previously loaded with ?OVLOD and frees the overlay area if the OUC becomes 0. You cannot issue ?OVREL from the overlay you want to release, because the normal return, in this case, would be in the target overlay.

Notes

- See the descriptions of ?OVLOD, ?OVEX, ?OVKIL in this chapter.

AOS/VS

?PCLASS *[packet address]*
error return
normal return

Input

AC0	Reserved (Set to 0.)
AC1	Reserved (Set to 0.)
AC2	Address of the ?PCLASS packet, unless you specify the address as an argument to ?PCLASS

Output

AC0	Unchanged
AC1	Unchanged
AC2	Address of the ?PCLASS packet

Error Codes in AC0

ERPRH	Attempt to access process not in hierarchy

Why Use It?

Use this system call to obtain locality and class information for a specific process.

Who Can Use It?

You must have Superprocess privilege or the target process must be in your hierarchy. There are no restrictions concerning file access.

What It Does

This system call accepts a process identifier from you. It returns the process's user locality, its program locality, and its class information.

If you want information about the calling process, place the value of ?PCL_PID in offset ?PCL_PKT.PCODE of the packet and place –1 in offset ?PCL_PKT.PID.

You obtain information about a different process in one of the two following ways:

- Assume that you know the name of the process. Place the value of ?PCL_PNAME in offset ?PCL_PKT.PCODE. Place in offset ?PCL_PKT.PNAME a byte pointer to the process name. Set offset ?PCL_PKT.PID to zero.
- Assume that you know the PID of the process. Place the value of ?PCL_PID in offset ?PCL_PKT.PCODE. Place zero in offset ?PCL_PKT.PNAME. Place the PID in offset ?PCL_PKT.PID.

Figure 2–175 shows the structure of the ?PCLASS parameter packet, and Table 2–154 describes its contents.

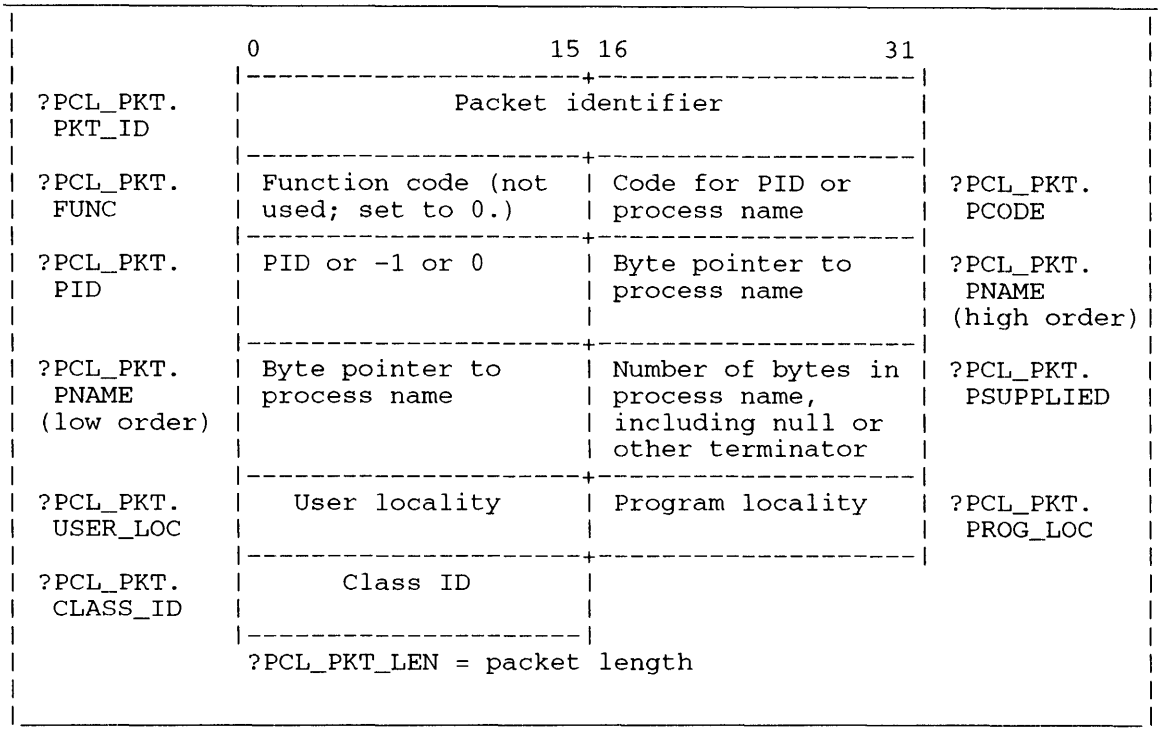


Figure 2–175. Structure of ?PCLASS Packet

Table 2–154. Contents of ?PCLASS Packet

Offset	Contents
?PCL_PKT.PKT_ID (doubleword)	Packet identifier. Place ?PCL_PKT_PKTID here.
?PCL_PKT.FUNC	Function code. Not used. (Set to 0.)
?PCL_PKT.PCODE	Code word into which you place ?PCL_PID when you supply a PID, or else ?PCL_PNAME when you supply a process name.
?PCL_PKT.PID	Supply -1 for the calling process. For another process, either place its PID here, or else place 0 here and the process's name information in the next two offsets.
?PCL_PKT.PNAME (doubleword)	If you've placed 0 in the previous offset, then place a byte pointer here to the process's name, and place its length in the next offset.
?PCL_PKT.PSUPPLIED	If you've placed 0 in offset ?PCL_PKT.PID and a byte pointer in the previous offset, then place the number of bytes in the byte pointer here.
?PCL_PKT.USER_LOC	User locality.
?PCL_PKT.PROG_LOC	Program locality.
?PCL_PKT.CLASS_ID	Class ID.

?PCNX — Passes a connection from one server to another in Ring 7.

?PCNX
error return
normal return

Input

AC0	Customer's PID
AC1	PID of the new server
AC2	Reserved (Set to 0.)

Output

AC0	Unchanged
AC1	Unchanged
AC2	Undefined

Error Codes in AC0

ERCBK	Connection broken
ERCDE	Connection does not exist
ERCCS	Cannot connect to self
ERNAS	Process is not a server
ERPRH	Attempt to access process not in hierarchy

Why Use It?

?PCNX allows you to break a connection between a server and one of its customers and to re–establish the connection with a new server process. ?PCNX is useful for passing a customer/server connection from an intermediate server to a specialized server.

Who Can Use It?

You need no special process privileges to issue this call. There are no restrictions concerning file access. However, the new server's PID must belong to a valid server process and also be a member of your process hierarchy. Since connections are between cooperating processes, the new server must cooperate with the customer process.

What It Does

?PCNX passes the connection established between the caller and one of its customers to another server process. Before you issue ?PCNX, load AC0 with the PID of the customer process, and load AC1 with the PID of the new server. Note that AC1 cannot contain the same PID number that is in AC0. This would connect the server to itself.

The calling process must be a server of the customer that you specify in AC0, and there must be an unbroken connection between the two. The new server process must have already issued ?SERVE to declare itself a server.

When a server issues ?PCNX, the operating system updates the connection–table entry so that it contains the PID of the new server.

?PIDS *[packet address]*
error return
normal return

Input		Output	
AC0	One of the following: • Host ID • –1 for the current host • Byte pointer to the hostname	AC0	Unchanged
AC1	Defines contents of AC0 as follows: • 0 if AC0 contains a host ID • –1 if AC0 contains a byte pointer • ignored if AC0 contains –1	AC1	Unchanged
AC2	Address of the ?PIDS packet, unless you specify the address as an argument to ?PIDS	AC2	Address of the ?PIDS packet

Error Codes in AC0

ERVWP	Invalid address passed as system call argument

Why Use It?

Use this system call to obtain information about the number of processes on your system.

Who Can Use It?

There are no special process privileges needed to issue this call, and there are no restrictions concerning file access. However, gaining access to a remote host requires that you have RMA access on the remote system. You obtain RMA access by first having a profile on the remote system with the same username and password as on your local system. Second, you must have privilege ?PRRAPRV in your profile on the remote system.

What It Does

You can issue ?PIDS to obtain the maximum number of PIDs allowed. ?PIDS also returns the number of processes currently running on the system and the number of processes below 256.

The process count limit might change in future revisions of the operating system. Furthermore, the number of processes allowed on the system is limited by a system generation question. Programs that allocate resources based on the maximum number of PIDs on a system need a way to obtain this limit; ?PIDS provides the way.

?PIDS Continued

Figure 2–176 shows the structure of ?PIDS's parameter packet, and Table 2–155 describes its contents.

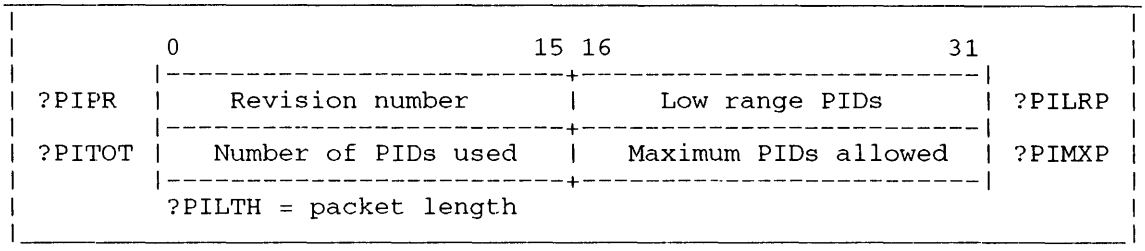


Figure 2–176. Structure of ?PIDS Packet

Table 2–155. Contents of ?PIDS Packet

Offset	Contents
?PIPR	Packet revision number. Place 0 here.
?PILRP	The operating system returns the number of PIDs used that are fewer than 256 (i.e., those in the low range).
?PITOT	The operating system returns the total number of PIDS that are in use. This includes the number of low-range PIDs that are in offset ?PILRP.
?PIMXP	The operating system returns the maximum number of PIDs allowed on the system. This value may differ from that of ?VSPIDS as defined in the parameter files.

?PMTPF
error return
normal return

Input		Output	
AC0	Reserved (Set to 0.)	AC0	Unchanged
AC1	Reserved (Set to 0.)	AC1	Unchanged
AC2	Packet address	AC2	Unchanged

Error Codes in AC0

ERAPU	Attempt to pass unheld access privileges
ERCBK	Connection has been broken
ERCDE	Connection does not exist
ERIFI	Invalid protected file ID
ERPRH	Process not in hierarchy
ERRNI	Ring number invalid

Why Use It?

The first opener of a file (?SOPPF) uses ?PMTPF to allow other segment images to open a protected shared file or to revoke this access. ?PMTPF provides you with a way to control access to protected shared files.

Who Can Use It?

The PID of the target segment must identify a process that is in your hierarchy. Also, read the explanation of granting access privileges in the following section "What It Does" and read the description of offset ?PFFLG in Table 2–155. The description of system call ?SOPPF contains the restrictions concerning file access.

What It Does

The first opener of a protected shared file uses ?PMTPF to grant access privileges to other segment images that wish to open the protected shared file. Only the first opener of a protected shared file can issue ?PMTPF against that file.

The ?PMTPF caller cannot grant any access privileges that it does not have itself. Also, ?PMTPF privileges are not cumulative. That is, the ?PMTPF caller grants a segment image only the access privileges that were specified in the last ?PMTPF call that was directed towards that segment image.

The first opener can revoke access privileges by issuing ?PMTPF with a null privileges mask.

A ?PMTPF access grant remains active until one of the following events occurs:

- The connection between the first opener and the target image is broken.
- The first opener closes the file.
- The first opener revokes the access grant.

NOTE: Revoking a ?PMTPF access grant affects only future opens.

To use ?PMTPF, you must supply a packet that contains various information about the target program. Figure 2–177 shows the structure of the packet and Table 2–156 describes its contents.

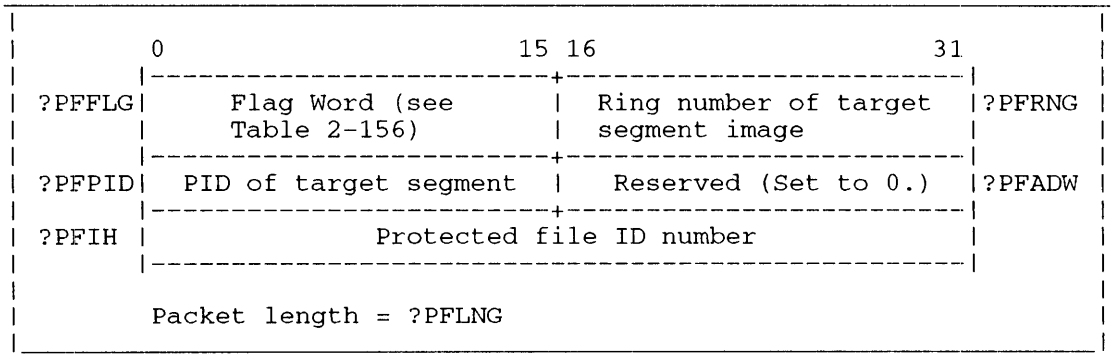


Figure 2–177. Structure of ?PMTPF Packet

Table 2–156. Contents of ?PMTPF Packet*

Offset	Contents
?PFFLG	Flag word ?FAC<O,W,A,R,E>--Access privileges. ?PFFO--First open. ?PFRW--Open for read and write.
?PFRNG	Ring number of target segment.
?PFPID	PID of target segment.
?PFADW	Reserved. (Set to 0.)
?PFIH (doubleword)	Protected file ID number.

* There is no default unless otherwise specified.

?PNAME — Gets a full process name.

?PNAME
error return
normal return

Input

AC0	Byte pointer to one of the following: • Buffer to receive the process name • Name of the target process, to obtain its PID
AC1	One of the following: • PID of the target process, to obtain its process name • 0 to obtain the PID of the target process • –1 to obtain the PID of the calling process
AC2	Reserved (Set to 0.)

Output

AC0	Unchanged
AC1	Requested PID
AC2	Undefined

Error Codes in AC0

ERPRE	Invalid or illegal system call parameter
ERPNM	Illegal process name
ERPRH	Attempt to access process not in hierarchy
ERVBP	Invalid byte pointer passed as a system call argument

Why Use It?

You can use ?PNAME to obtain both the process name and PID of a process. Therefore, ?PNAME is a useful complement to system calls that require these identifiers as input.

Who Can Use It?

There are no special process privileges needed to issue this call, and there are no restrictions concerning file access.

?PNAME Continued

What It Does

?PNAME returns either the PID or the process name of a target process, depending on your input to AC0 and AC1.

To obtain the name of a process other than the calling process, load AC1 with the PID of that process, and load AC0 with a byte pointer to a buffer that will receive the process name. To obtain the PID of a process (except the calling process), load AC1 with 0, and load AC0 with a byte pointer to the process's name. The operating system returns the PID you requested to AC1.

To obtain the PID of the calling process, load AC1 with –1 and AC0 with 0. The operating system returns the caller's PID to the same accumulator. To obtain the name of the calling process, load AC0 with a byte pointer to a receive buffer.

?PRCNX Passes a connection from one server to another.

?PRCNX
error return
normal return

Input

AC0	Customer's PID
AC1	PID of the new server
AC2	A field that describes the following: • Bits 0 through 20 are reserved (Set to 0.) • Bits 21 through 23 contain the ring of the customer • Bits 24 through 28 are reserved (Set to 0.) • Bits 29 through 31 contain the ring of the specified server

Output

AC0	Unchanged
AC1	Unchanged
AC2	Unchanged

Error Codes in AC0

ERCBK	Connection broken
ERCDE	Connection does not exist
ERCCS	Cannot connect to self
ERNAS	Process is not a server
ERPRH	Attempt to access process not in hierarchy
ERRNI	Invalid ring number

Why Use It?

?PRCNX allows you to break a connection between a ring of a server and one of its customers and to re–establish the connection with a ring of a new server process. ?PRCNX is useful for passing a customer/server connection from an intermediate server to a specialized server.

Who Can Use It?

The calling process must be a server of the customer that you specify in AC0 and AC2, and there must be an unbroken connection between the two. The new server process must have already issued ?SERVE to declare itself a server. There are no restrictions concerning file access.

What It Does

?PRCNX passes the connection established between the calling segment image (PID/ring tandem) and one of its customers to a segment image of another server process. Before you issue ?PRCNX, load AC0 with the PID of the customer process, and load AC1 with the PID of the new server. Note that AC1 cannot contain the same PID number that is in AC0. This would connect the server to itself.

?PRCNX Continued

When a server issues ?PRCNX, the operating system updates the connection–table entry so that it contains the PID/ring of the new server and deletes any previous connection between the customer and the new server. In effect, this breaks the old connection. Therefore, the operating system revokes any protected shared file access grants that the old server allowed the customer.

Notes

- See the description of ?PCNX in this chapter.

?PRDB *[packet address]*
error return
normal return

?PWRB *[packet address]*
error return
normal return

Input

AC0	Reserved (Set to 0.)
AC1	Target file's channel number
AC2	Address of the packet, unless you specify the address as an argument to the system call

Output

AC0	Undefined
AC1	Byte count of the bytes read or written
AC2	Address of the packet

Error Codes in AC0

ERVWP Invalid word pointer passed as a system call argument

ER_FS_DIRECTORY_NOT_AVAILABLE
Directory not available because the LDU was force released (AOS/VS II only)

ER_FS_TLA_MODIFY_VIOLATION
Attempt to modify an AOS/VS II file with ?ODTL value supplied in ?GOPEN packet (?PWRB only)

Why Use It?

You can use ?PRDB or ?PWRB to check for bad blocks on a disk, or to check for problems with an I/O device.

When the operating system encounters a transfer error (bad block) while it is executing a ?PRDB or a ?PWRB, it takes the normal return and reports the reason for the error in the packet, but transfers all or part of that bad block. (See Table 2–159.) However, when a device error occurs during a ?PRDB or a ?PWRB, the operating system immediately aborts the transfer and returns the device error code to the packet.

Who Can Use It?

There are no special process privileges needed to issue this call. You must have obtained a channel number via ?GOPEN or ?OPEN before issuing this call. Also, you have Read access to the file before issuing ?PRDB or Write access to the file before issuing ?PWRB.

What It Does

?PRDB and ?PRWB read and write physical blocks on disk, respectively. Before you issue one of these system calls, load AC1 with the channel number assigned to the target file at ?GOPEN time.

?PRDB and ?PWRB each require an extended packet. You can specify the packet address as an argument to the system call or as input to AC2 before you issue the system call. The parametric value for the length of the packet, including the extension, is ?PPBLT. Figure 2–178 shows the structure and Table 2–157 describes the contents of the ?PRDB/?PRWB packet. Table 2–158 lists the disk and tape types applicable to the offsets in the packet extension.

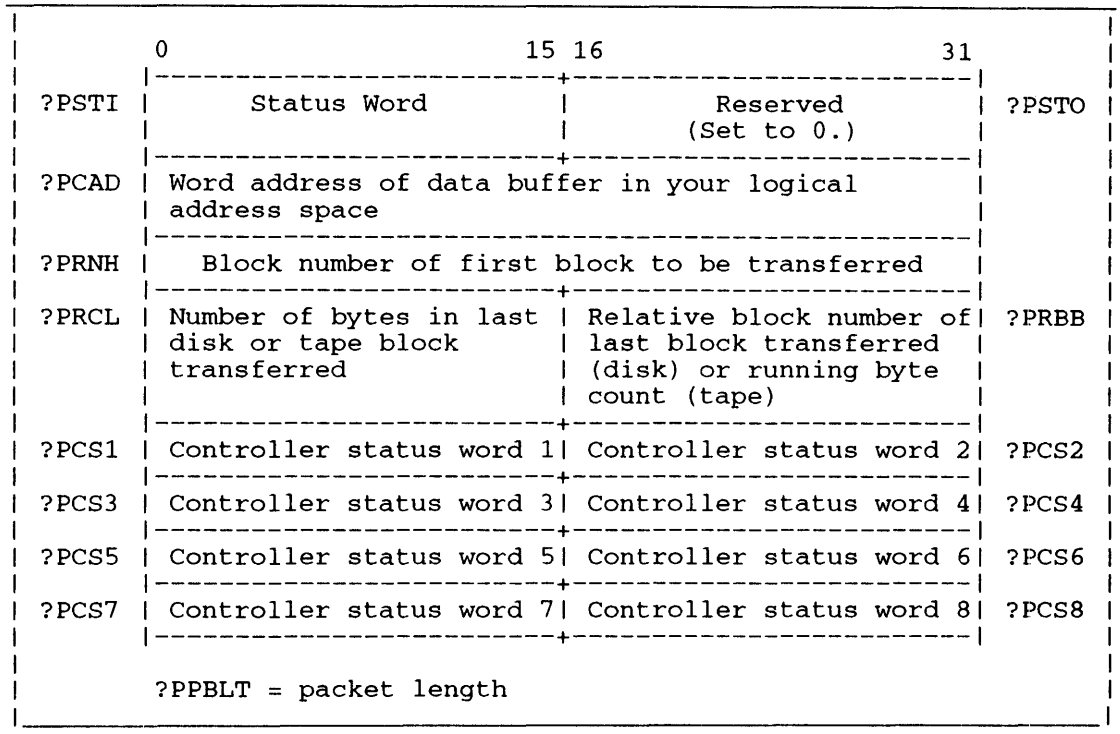


Figure 2–178. Structure of ?PRDB / ?PWRB Packet

?PRDB/?PRWB Packet Offsets

Specify the tape or printing control options you want in the left byte of ?PSTI. Specify the number of blocks you want to read or write in the right byte of offset ?PSTI (block count). Use offset ?PCAD as a word pointer to the address of the data buffer you have reserved in your logical address space for the transfer. Offset ?PRNH must indicate the relative block number of the first block you want to transfer.

Use offset ?PRCL for ?PWRB (write block) only. Offset ?PRCL must contain the number of bytes in the last block you are writing. In effect, this value specifies the last valid byte in the block. If you are performing a write (?PWRB) to extend the file, the operating system places the end–of–file mark immediately after this byte. If you set offset ?PRCL to 0, the operating system sets the bytes in the last block to the default, which is 512 (a full block).

As Table 2–157 and Table 2–158 indicate, the operating system uses the ?PRDB/?PRWB packet extension to return status information about the block transfer. (You supply input values only for offsets ?PSTI through ?PRCL in the main packet.)

Table 2–157. Contents of ?PRDB/?PWRB Packet

Offset	Input Value	Output Value
?PSTI	Left Byte: Input options. ?IMIO = 1B(?IDIO) Use direct I/O with the MCA Protocol. The system ignores the block count, and transmits the number of bytes in ?PRCL. ?ENOV = 1B(?ENOR) For printers, enable a Vertical Form Unit load (part of Forms Control Utility). For tape, override the LEOT (logical end-of-tape) mark. ?SAFM = 1B(?SAFE) Enable a safe write to tape.	Unchanged.
	Right Byte: Block count.	Unchanged.
?PSTO	Reserved.	Unchanged.
?PCAD (doubleword)	Address of data buffer in your logical address space.	Unchanged.
?PRNH (doubleword)	Block number of first block to be transferred.	Unchanged.
?PRCL	Number of bytes in last disk or tape block transferred.	Unchanged.
?PRBB	Not applicable.	Number of blocks successfully transferred (disk) or running byte count (tape).
?PCS1 - ?PCS8	Not applicable.	Controller status words. (See Table 2-158.)

You can use offset ?PRBB to determine if an error was encountered, and if one was, the block number of the first bad block encountered during the transfer. Offset ?PRBB records the number of blocks that were successfully transferred. If ?PRBB does not contain the number you specified in offset ?PSTI, an error has occurred and offsets ?PCS1 through ?PCS8 in the packet extension record the reason for the transfer error.

Offsets ?PCS1 and ?PCS5 are always set on output, as appropriate for the device in use. Note that offsets ?PRBB through ?PCS5 are the only packet extension parameters currently in use.

Table 2–158. ?PRDB/?PWRB Packet: Controller Status Words

DPD/DPG Disk	Status Word	Use
6030 (Floppy) 6045/6050/6051 (10 Mbytes) 6070 (20 Mbytes)	?PCS1	DIA (Transfer) Status
DPI Disk	**Status Word**	**Use**
6097 (Floppy) 6098/6099 (12.5 Mbytes) 6100/6103 (25 Mbytes) 6225 (5 Mbytes) 6227 (15 Mbytes) 6234 (50 Mbytes)	?PCS1	DIA (Transfer) Status
DKB Disk	**Status Word**	**Use**
6063/6064/6065/6066 (Fixed Head)	?PCS1 ?PCS2 ?PCS3 ?PCS4	DIA (Transfer) Status Unused ERCC Word One ERCC Word Two
DPF Disk	**Status Word**	**Use**
6060 (96 Mbytes) 6061 (147 Mbytes) 6067 (50 Mbytes) 6122 (277 Mbytes) 6160 (73 Mbytes) 6161 (147 Mbytes) 6214 (602 M bytes)	?PCS1 ?PCS2 ?PCS3 ?PCS4	DIA (Transfer) Status DIB (Drive) Status ERCC Word One ERCC Word Two
DPM Disk	**Status Word**	**Use**
4514 (5 1/4 in. Floppy)	?PCS1	DIA (Transfer) Status
DPJ Disk	**Status Word**	**Use**
6236/6237 (354 Mbytes) 6239/6240/ 6290/6350 (592 Mbytes) 6309 (5 1/4 in. Floppy) 6310 (38 Mbytes) 6328 (70 Mbytes) 6329 (120 Mbytes)	?PCS1 ?PCS2 ?PCS3 ?PCS4 ?PCS5	CB Status Unit Status CB Error Status Disk Error Code Controller Type

(continued)

Table 2–158. ?PRDB/?PWRB Packet: Controller Status Words

MTB/MTD Tape	Status Word	Use
6026 (800/1600 bpi) 4307 (1600/6250 bpi) 6200 (1600/6250 bpi) 6300 (1600/6250 bpi)	?PCS1 ?PCS2	DIA (Transfer) Status DIC (Drive) Status
MTC Tape	**Status Word**	**Use**
6125 (1600 bpi) 6231/6311 (Cartridge)	?PCS1	DIA (Transfer) Status
MRC Disk	**Status Word**	**Use**
6236/6237 (354 Mbytes) 6239/6240/6290 (592 Mbytes) 6357/6398/ 6399/6400 (862 Mbytes) 6581/6582/6584 (500 Mbytes)	?PCS1 ?PCS2 ?PCS3 ?PCS4 ?PCS5 ?PCS6	Controller Type Unit Type Controller Error Code Error Status (ctrl/unit) Unit Status Drive Error Code
MRC Tape	**Status Word**	**Use**
6299/6300 (6250/1600 bpi) 4307-TL (6250/1600/800 bpi)	?PCS1 ?PCS2 ?PCS3 ?PCS4 ?PCS5 ?PCS6	Controller Type Unit Type Controller Error Code Error Status (ctrl/unit) Unit Status Drive Error Code

(concluded)

Table 2–159. Error Reports Returned in ?PRDB/?PWRB Offsets

Offsets	Contents	Meaning
?PSTI ?PRBB	20 20	The operating system transferred all blocks without error.
?PSTI ?PRBB	20 19	Last block was not transferred successfully. You might want to try again.

Notes

- Refer to the *ECLIPSE® MV/Family (32-Bit) Systems Principles of Operation* manual for details on the DIA, DIB, and DIC I/O instructions. Or, refer to the programmer's reference manual for your particular model.

?PRI
error return
normal return

Input

AC0	Priority number you wish to assign to the calling task
AC1	Reserved (Set to 0.)
AC2	Reserved (Set to 0.)

Output

AC0	Unchanged
AC1	Undefined
AC2	Undefined

Error Codes in AC0

No error codes are currently defined.

Why Use It?

?PRI is analogous to ?IDPRI, except that it changes the priority of the calling task, rather than another target task. Like ?IDPRI, ?PRI gives you some control over the operating system's task–scheduling activities. For example, you might use ?PRI to favor the calling task by assigning it a higher relative priority or, conversely, to favor other tasks by giving the calling task a lower relative priority.

Who Can Use It?

There are no special process privileges needed to issue this call, and there are no restrictions concerning file access.

What It Does

?PRI changes the priority number of the calling task to the priority that you specify in AC0. If there are already other tasks at the new priority level, the calling task ranks last in the priority queue and, therefore, gains CPU access after all others at that priority level. Like ?IDPRI, ?PRI can cause immediate task rescheduling.

Before you issue ?PRI, load AC0 with the priority number that you want to assign to the calling task. Priority numbers for tasks range from 0 (the highest priority level) through 255 (the lowest priority level). If you specify a priority number greater than 255, the operating system truncates the priority number to its least significant 8 bits.

?PRIPR
error return
normal return

Input

AC0 One of the following:

- –1 to change the priority of the calling process
- Byte pointer to the name of the target process
- PID of the target process

AC1 One of the following:

- –1 if AC0 contains a byte pointer
- 0 if AC0 contains a PID

Otherwise, ignores contents of AC1.

AC2 New priority of the target process in the following ranges:

- 1 through 255 for resident and pre–emptible processes
- 1 through 3 for swappable processes

Output

AC0 Unchanged

AC1 Unchanged

AC2 Unchanged

Error Codes in AC0

ERMPR	System call parameter address error
ERPNM	Illegal process name
ERPRH	Attempt to access process not in hierarchy
ERPRP	Illegal priority (Either a caller with privilege ?PVPR tried to raise its own priority or the newly assigned priority is out of range.)
ERVBP	Invalid byte pointer passed as a system call argument

Why Use It?

?PRIPR changes a process's priority. Process priority is one of several criteria the operating system uses to determine the scheduling order for individual processes.

?PRIPR Continued

You can use ?PRIPR to favor a process for CPU access (by raising its priority) or to favor other processes over the target process (by lowering the target's priority).

Who Can Use It?

The ?PRIPR call requires no special process privileges. A process can issue ?PRIPR to change the priority of any subordinate process. However, if the calling process is in Superprocess mode, it can change the priority of any process. There are no restrictions concerning file access.

What It Does

?PRIPR changes the priority of the process that you specify in AC0 (the calling process or another target process).

The priority range for all processes is 1 (the highest priority) through 511 (the lowest priority).

Notes

- See the description of ?CTYPE in this chapter.

?PRKIL — Kills all tasks of a specified priority.

?PRKIL
error return
normal return

Input		Output	
AC0	Priority level of the tasks to kill	AC0	Unchanged
AC1	Reserved (Set to 0.)	AC1	Undefined
AC2	Reserved (Set to 0.)	AC2	Undefined

Error Codes in AC0

No error codes are currently defined.

Why Use It?

?PRKIL allows you to kill all tasks of a given priority level. (You can group tasks of similar functions under the same priority.)

Who Can Use It?

There are no special process privileges needed to issue this call, and there are no restrictions concerning file access.

What It Does

?PRKIL kills (terminates) all tasks of the priority level that you specify in AC0. If the target tasks were explicitly suspended by ?SUS, ?IDSUS, or ?PRSUS, or because they issued an ?XMTW or ?REC, the operating system lifts the suspension and readies the target tasks at the highest priority level (0) before it executes the task–kill logic. If the target tasks became suspended by issuing other system calls, the operating system aborts those outstanding system calls and readies the tasks before it performs the task–kill logic.

If you supplied a kill–processing routine for the target tasks, the operating system passes control to those routines when you issue ?PRKIL. You should end each kill–processing routine with a ?KILL. ?KILL invokes a ?UKIL task termination routine — either the system default ?UKIL or a ?UKIL you have defined. ?UKIL, and then terminates the tasks.

If you did not supply kill–processing routines, control passes to the appropriate ?UKIL routine immediately.

Notes

- See the descriptions of ?IDSUS, ?PRSUS, ?XMTW, ?REC, and ?KILL in this chapter.

?PROC *[packet address]*
error return
normal return

Operating System Differences	
Accumulator	
Input and Output	None
Error Codes	Some
Parameter Packet	Some

Input

AC0	Reserved (Set to 0.)
AC1	Reserved (Set to 0.)
AC2	Address of the ?PROC packet, unless you specify the address as an argument to ?PROC

Output

AC0	Undefined
AC1	PID of the new process
AC2	Address of the ?PROC packet

Error Codes in AC0

ERBMX	Illegal maximum process size on process create
ERCON	Terminal device specification error
ERDAD	Directory access denied (AOS/VS only)
ERDPT	Different process type
EREAD	Execute access denied (You must have Execute access to the program file you want to run.) (AOS/VS only)
EREXC	Resident process tried to ?PROC and block
ERIFL	IAC (Intelligent Asynchronous Controller) failure
ERIFT	Illegal file type
ERPDF	Error in process UST definition
ERPRN	Number of processes exceeds maximum
ERPRP	Illegal priority (The caller tried to assign the new process a higher priority than its own, without the privilege to do so.) (AOS/VS only)
ERPRV	Caller not privileged for this action (AOS/VS only)
ERPTY	Illegal process type (The caller tried to assign the new process a process type different from its own, without the privilege to do so.) (AOS/VS only)
ERSMX	Error in setting maximum CPU time
ERSNM	Attempt to exceed maximum number of sons (AOS/VS only)
ERUNM	Illegal username
ERVBP	Invalid byte pointer passed as a system call argument
ERVWP	Invalid word pointer passed as a system call argument
ERWSM	Attempt to set working set minimum, not privileged
ERWSS	Invalid working set maximum/minimum (AOS/VS only)

Why Use It?

?PROC allows you to create a process and define its privileges and characteristics.

Who Can Use It?

Although there are no special process privileges needed to issue this call, many options affect the privileges the newly created process has. For example, setting bit ?PFDA in offset ?PFLG prevents the operating system from passing the father's default ACL to the son. These options appear throughout the rest of this explanation of ?PROC.

You must have Execute access to the directory in which the program file resides along with both Execute and Read access to the program file itself.

What It Does

?PROC creates a process and assigns it whatever characteristics you specify in the ?PROC packet. Before you issue ?PROC, set up a packet in your logical address space that is ?PLTH words long. You can either cite the packet address as an argument to ?PROC, or you can load the packet address into AC2 before you issue ?PROC. Figure 2–179 shows the structure of the ?PROC packet and Table 2–160 defines each offset and mask.

Among other characteristics, ?PROC defines the process's type and priority, its maximum and minimum working set size, and its creation privileges (that is, the right to create sons with additional ?PROC system calls). Note that you cannot create a son process with privileges the calling process does not have.

Note also that when the operating system terminates a father process, it simultaneously terminates all his sons.

The ?PROC extension packets let you specify the following items when you issue ?PROC:

- CPU selection.
- User locality.
- ?PROC–related username.
- ?GROUP access control list (AOS/VS II only).

Figures 2–180 and 2–181 show the structure of the ?PROC extension packets, and Tables 2–162 and 2–163 define each offset.

Offset ?PFLG

Offset ?PFLG in the packet contains specification bits for the following characteristics and privileges:

- Process type.

  Bit ?PFRS assigns the new process resident status; bit ?PFRP assigns it pre–emptible status. (Swappable is the default process type.) Bits ?PFRS and ?PFRP are mutually exclusive; if you set them both, the operating system takes the ?PROC error return and passes error code ERPTY to AC0.

- Default access control (AOS/VS only).

  Bit ?PFDA, the access privileges bit, prevents the operating system from passing the father's file access privileges to the son. But, bit ?PFDA is only checked by the operating system if a username is specified, and if ?PUNM does *not* contain –1. If you set ?PFDA, the operating system assigns the son the default access control list USERNAME, OWARE (Owner, Write, Append, Read, Execute access). The son can also receive USERNAME OWARE access under the following conditions:

  – When you assign the son a different username from its father.

  – When the father process has no default access privileges.

  For more information on default file access, see the description of ?DACL in this chapter.

- Control directive.

  Bit ?PFDB directs the operating system to pass control to the Debugger utility immediately after it creates the new process. By default, control passes to the start of the program associated with the new process.

- Concurrency.

  Bit ?PFEX blocks the calling process while the new process executes. If you don't set this bit, the son runs concurrently with its father. The calling process must have privilege ?PVEX to allow this. Under AOS/RT32, all processes have this privilege.

- Privileges mask.

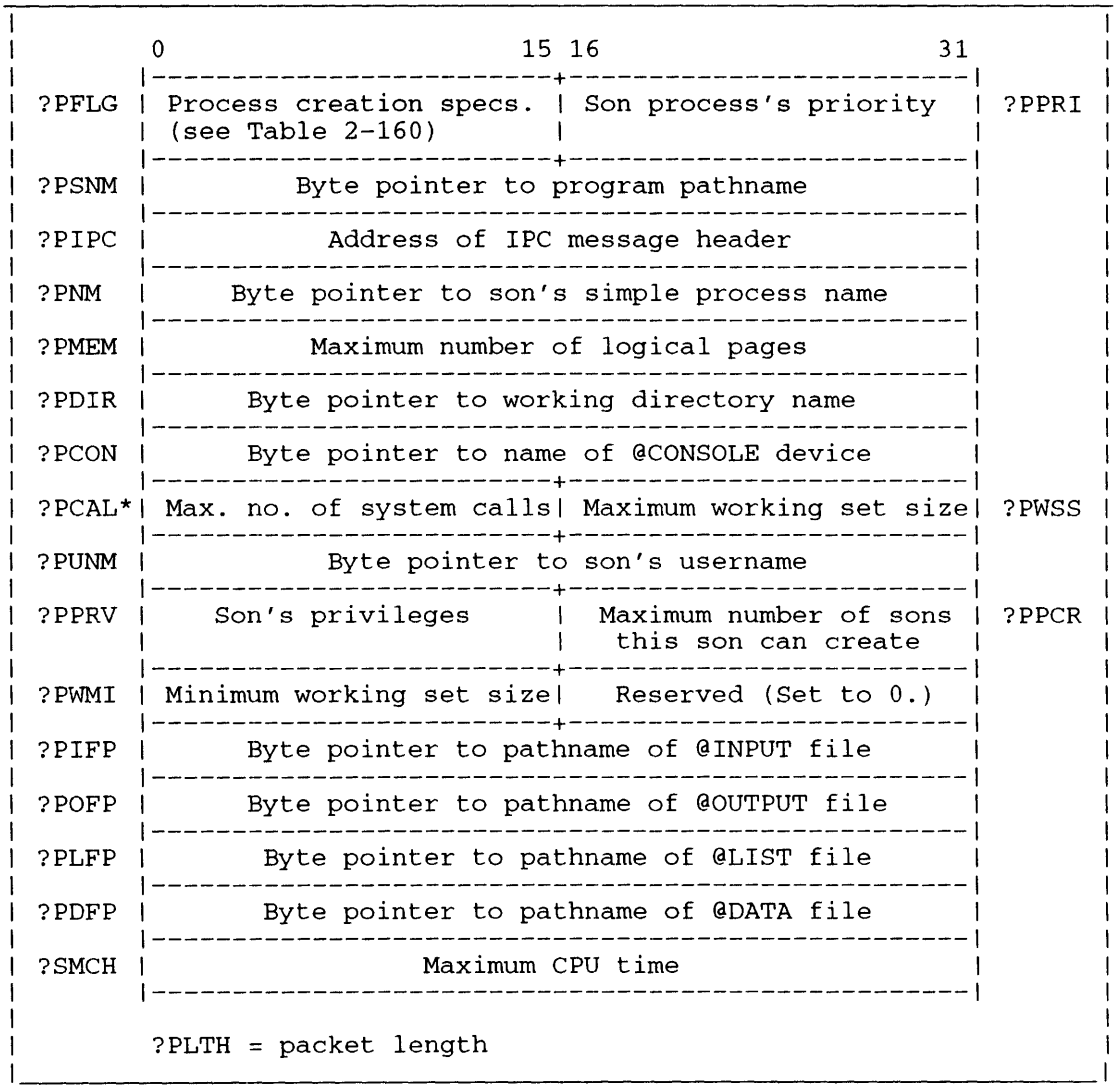
  Bit ?PFPM, the privileges mask, directs the operating system to give the son all the father's privileges except those specified in offset ?PPRV. Alternatively, if you set ?PFPM and set ?PPRV to –1, the son gets no privileges. If you just want the son to inherit the father's privileges, do NOT set ?PFPM, but set ?PPRV to –1.

- Break files and memory dumps.

  On an AOS/VS system, setting ?PBRK and not ?PDMP creates a break file if the process traps. Not setting ?PBRK and setting ?PDMP creates a dump file of Ring 7 if the process traps. Setting both bits has exactly the same effect as not setting ?PBRK and setting ?PDMP.

  On an AOS/RT32 system, all of the three combinations in the previous paragraph have the same effect. The effect is to create one break file with a memory dump of Ring 7. The filename has the format ?PID.TIME.7.BRK where PID and TIME are as described in Table 2–13 under offset ?ENFNP. The operating system creates another break file for each additional user ring (4, 5, 6) that is defined. A possible filename in this additional case is ?00035.11_41_23.6.BRK.

Table 2–161 describes each of the privilege bits in offset ?PPRV.



* This offset differs between AOS/VS and AOS/RT32. See Table 2-160.

Figure 2-179. Structure of ?PROC Packet

Table 2–160. Contents of ?PROC Packet*

Offset	Contents
?PFLG	Process creation specifications. ?PFPP--Reserved - set to 0. Control directive. ?PFDB--Control goes to the Debugger. DEFAULT = 0 (control goes to the starting address of the program file). Concurrency. ?PFEX--Caller is blocked while son executes. DEFAULT = 0 (son runs concurrently with the ?PROC caller). Privileges mask. ?PFPM--Mask son's privileges. DEFAULT = 0 (son process receives all privileges selected in ?PPRV). Access privileges. ?PFDA--Do not pass default ACL. ?PFDA is only checked if a username is specified, and ?PUNM does not contain -1. DEFAULT = 0 (son has same default ACL as father (the caller)). Break file. ?PBRK--Create a break file if process traps or terminates fatally. DEFAULT = 0 (do not create a break file if the process traps or terminates fatally). Block the son process. ?PFBS--Block son. DEFAULT = 0 (do not block the son). Extension packet. ?PFXP--An extension packet exists. DEFAULT = 0 (no extension packet exists).

* There is no default unless otherwise specified. (continued)

Table 2–160. Contents of ?PROC Packet*

Offset	Contents
?PFLG (continued)	Process type ?PFRP--pre-emptible. ?PFRS--resident. DEFAULT = 0 (son is a swappable process).
	Process traps. ?PDMP--Dumps Ring 7. (If ?PBRK is also set, overrides ?PBRK.)
?PPRI	Priority of the son process. DEFAULT = -1 (same as caller's priority).
?PSNM (doubleword)	Byte pointer to pathname of program file for the new process to execute.
?PIPC (doubleword)	Address of an IPC message header; this message is sent to the new process. DEFAULT = -1 (no message header).
?PNM (doubleword)	Byte pointer to new process's simple process name. DEFAULT = -1 (Son's simple process name is the ASCII representation of its PID (five digits, including leading zeros).)
?PMEM (doubleword)	Maximum number of logical pages in new process. DEFAULT = -1 (same logical page maximum as the caller's).
?PDIR (doubleword)	Byte pointer to name of son's working directory; if 0, then use the caller's current working directory. DEFAULT = -1 (same initial working directory as the caller's).

* There is no default unless otherwise specified. (continued)

Table 2–160. Contents of ?PROC Packet*

Offset	Contents
?PCON (doubleword)	Byte pointer to name of new process's @CONSOLE file; if 0, there is no @CONSOLE file. DEFAULT = -1 (same @CONSOLE device as caller).
?PCAL	Maximum number of system calls the son can issue concurrently. DEFAULT = -1. (Son is limited to two concurrent system calls.) AOS/RT32 ignores this offset; set to -1.
?PWSS	Maximum working set size for son. DEFAULT = -1 (no limit on working set size; otherwise, caller's limit). NOTE: The calling process must have the ?PVWS privilege to set this parameter.
?PUNM (doubleword)	Byte pointer to son's username. DEFAULT = -1 (same username as the caller's). NOTE: If ?PUNM contains -1 and a username is specified, ?PFDA is also checked.
?PPRV	Son's privileges; each bit in this offset (Table 2-161) assigns a privilege if ?PFPM is not set, or denies a privilege if ?PFPM is set. ?PFPM is a flag in offset ?PFLG (described earlier in this table). DEFAULT = -1. (If ?PFPM is also set, son has no privileges; if ?PFPM is not set, son has all the caller's privileges.)
?PPCR	Maximum number of sons this son can create; if 0, no sons. DEFAULT = -1 (same number of sons as the caller's (minus the number already created)).

* There is no default unless otherwise specified. (continued)

Table 2–160. Contents of ?PROC Packet*

Offset	Contents
?PWMI	Minimum working set size. DEFAULT = -1 (no working set minimum; otherwise, caller's minimum). NOTE: The calling process must have ?PVWS privilege to set this parameter.
?PIFP (doubleword)	Byte pointer to pathname of son's @INPUT file; if 0, there is no @INPUT file. DEFAULT = -1 (same @INPUT file as the caller).
?POFP (doubleword)	Byte pointer to pathname of son's @OUTPUT file; if 0, there is no @OUTPUT file. DEFAULT = -1 (same @OUTPUT file as the caller).
?PLFP (doubleword)	Byte pointer to pathname of son's @LIST file; if 0, there is no @LIST file. DEFAULT = -1 (same @LIST file as the caller).
?PDFP (doubleword)	Byte pointer to pathname of son's @DATA file; if 0, there is no @DATA file. DEFAULT = -1 (same @DATA file as the caller).
?SMCH (doubleword)	Maximum CPU time allotted to the son. DEFAULT = -1 (Son receives remainder of father's time limit); if -1 and if father has no time limit, son has no time limit.

* There is no default unless otherwise specified. (concluded)

Table 2–161 describes each of the privilege bits in offset ?PPRV.

Table 2–161. Privilege Bits in Offset ?PPRV

Privilege	Meaning
?PVPC	The new process can create an unlimited number of sons.
?PVWS	The new process can ?PROC sons of a different program file type (that is, 16-bit or 32-bit program file).
?PVEX	The new process can remain unblocked while its sons execute.
?PVWM	The new process can define its sons' working set limit.
?PVPR	The new process can either assign itself a higher priority by issuing ?PRIPR or it can ?PROC a son with a higher priority than itself.
?PVTY	The new process can change its own process type (?CTYPE) or ?PROC a son with a different process type. (By default, sons have the same process types as their fathers.)
?PVUI	The new process can assign its sons different usernames.
?PVDV	The new process can define and access user devices.
?PVIP	The new process can issue the primitive IPC system calls ?ISEND and ?IS.R. (Note that connected processes, i.e., customer/servers, do not need this privilege to issue ?IS.R.)
?PVSM	The process can become System Manager.
?PVSU	The new process can issue ?SUSER to enter Superuser mode.
?PVSP	The new process can issue ?SUPROC to enter Superprocess mode.
?PVPP	The new process can be a peripheral process.

Offset ?PIPC

Offset ?PIPC points to an IPC message header. Frequently, you may want to send a CLI–format command line as the IPC message. This way, the son process can access arguments and switch values via a ?GTMES call. The explanation of system call ?GTMES explains a CLI–format line. In Appendix A sample programs BOOMER, DLIST, and TIMEOUT illustrate the ?GTMES call. Here's how to create such an IPC message:

- Set the system flag word (offset ?ISFL) to zero in the IPC message (as opposed to setting a bit if you were chaining to another process).
- Set bit ?RFCF in the user flag offset (?IUFL) to indicate CLI format.
- Set destination and origin port numbers to 0.
- Set offset ?ILTH to the word length of the message.
- Set offset ?IPTR to the word address of the message.

See Figure 2–83 for the structure of the ?ISEND header.

Suppose you issue ?PROC with offset ?PIPC containing the address of an IPC message header, and the target process will read the IPC message via ?GTMES. Then, offset ?IUFL of the IPC message header must contain ?RFCF to specify CLI format. Figure 2–83 and Table 2–60 describe the IPC message header; Figure 2–82 describes offset ?IUFL.

Other Offsets

Offset ?PPRI specifies a priority number for the new process, within the range 1 through 511. When you default this parameter (set offset ?PPRI to −1), the new process assumes its father's priority number. A father process must have privilege ?PVPR to assign to its son a higher priority than its own.

Offset ?PMEM specifies the maximum number of logical pages the new process can contain. By default, fathers and sons have the same logical page limit.

Offset ?PCAL specifies the maximum number of concurrent system calls the new process can issue. In general, critical processes should have higher ?PCAL parameters than less critical processes. Note that ?PCAL, and all specifications in ?PROC, should match the new process's needs as closely as possible.

Creating Offspring

The system determines the number of offspring that a process can create by checking its ?PROC packet for the following:

- The ?PVPC privilege in offset ?PPRV. This privilege allows the new process to create an unlimited number of sons. ?PVPC overrides the value in offset ?PPCR, described next.
- The ?PPCR offset which specifies the maximum number of offspring the new process can create. This offset is cumulative, which means that if a process with a ?PPCR value of 10 creates 2 sons, each with a ?PPCR value of 4, the original process cannot create any more sons, because 2 sons plus 8 (2*4) potential grandsons equals 10. This example is true as long as the 2 sons actually create 8 processes. If they actually create only 5 processes between them, the original process can create at most 3 processes (2 + 5 + 3 = 10).

- The ?PVEX bit in offset ?PPRV and the ?PFEX mask in offset ?PFLG. ?PVEX ensures that the new process remains unblocked while its sons execute (which allows it to create other sons). ?PFEX results in the ?PROC caller being blocked while the new process executes.

When a process which lacks the ?PVPC privilege tries to create a son, the operating system will create the son if

- The number of sons and their combined ?PPCR count (maximum number of sons which can be created) is less than or equal to the caller's ?PCPR value. (If the sum is more, the operating system signals an error.)
- Bit ?PVEX is set. (This lets the new process remain unblocked while its sons execute.)

Setting Maximum CPU Time

Offset ?SMCH specifies the maximum number of CPU milliseconds you want to assign the new process and all its subordinates. You must express this parameter as a double–precision unsigned integer. When a process exceeds its CPU time limit, the operating system terminates that process and returns error code ERTLM to its father.

When a process with a CPU time limit creates a son process, the operating system blocks the father and limits the son to the father's remaining CPU time. You can assign the son less than the father's remaining CPU time, but not more. If you assign more, the system takes the ?PROC error return and returns error code ERSMX (error in setting CPU time). As soon as the son terminates, the father receives whatever CPU time remains.

If you set offset ?SMCH to −1, the son acquires the rest of the father's CPU time. If you set offset ?SMCH to 0, the son has no CPU time limit, unless the father has a time limit. If the father has a time limit, the son gets the father's remaining time.

Setting the Working Set Size

Offsets ?PWMI and ?PWSS specify the new process's minimum and maximum working set size, respectively. Both values relate to the number of physical pages the process needs to execute efficiently. To set these values, the calling process must have privilege ?PVWS.

By default, the operating system adjusts the working set parameters dynamically, based on the process's type, page fault history, and general system overhead. When you set ?PWMI and ?PWSS, you direct the operating system to keep the working set within specific limits.

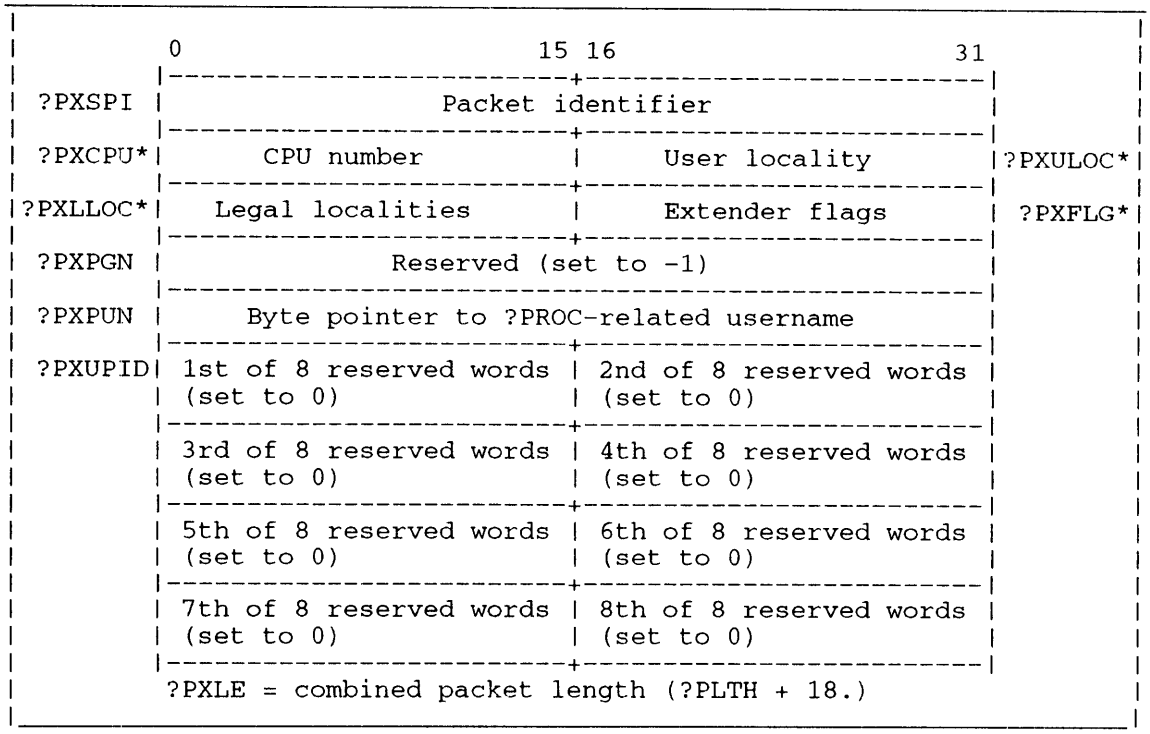
A process's working set size does not necessarily relate to the size of its logical address space. In fact, a larger process can require fewer physical pages than a smaller one if the larger process localizes its references and uses its databases efficiently. If you do set the working set size, take care not to set the maximum too low, because doing so increases paging I/O and processing time. Also, take care not to set the minimum working set size too high, because doing so can reduce efficient memory usage throughout your system.

CAUTION: *Generally, you will hurt the performance of the system if you define the size of a process's working set. Use this option with care; most of the time you'll be better off taking the default and allowing the operating system to make the necessary adjustments.*

Extension Packet

You specify the existence of an optional extension packet by setting bit ?PFXP in offset ?PFLG of the main parameter packet. Then place the extension packet itself immediately after the end of the main parameter packet. The length of the combined packet is ?PXLE words.

Figure 2–180 contains the structure of the ?PROC extension packet for AOS/VS and AOS/RT32 and Table 2–162 describes its contents. Figure 2–181 contains the structure of the ?PROC extension packet for AOS/VS II and Table 2–163 describes its contents.



* This offset differs between AOS/VS and AOS/RT32. See Table 2-162.

Figure 2-180. Structure of ?PROC Extension Packet for AOS/VS and AOS/RT32

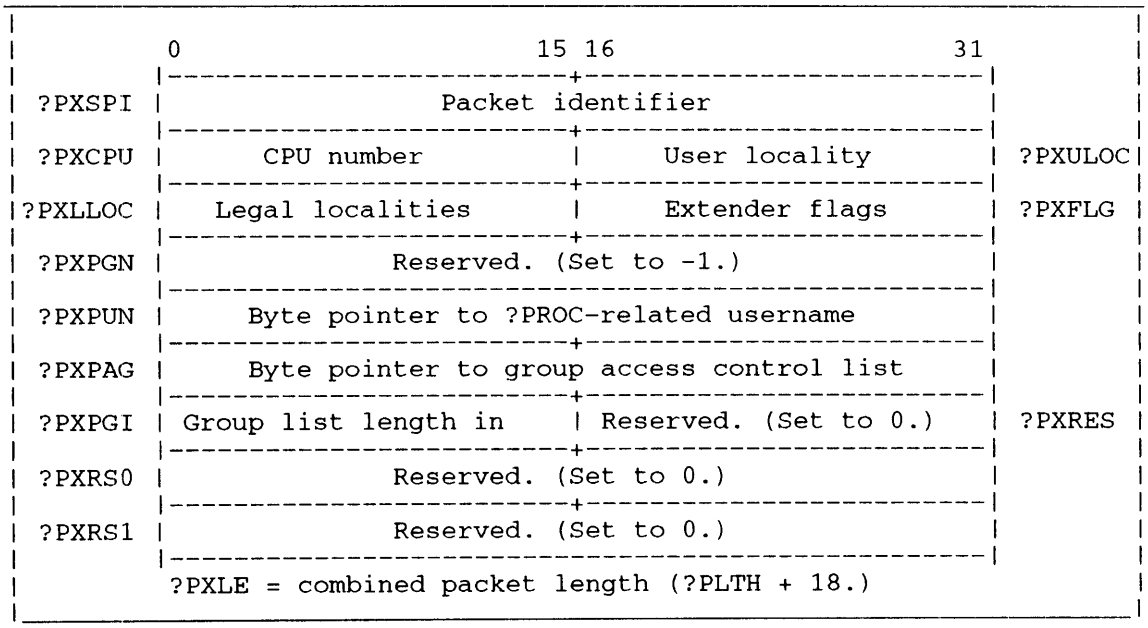


Figure 2-181. Structure of ?PROC Extension Packet for AOS/VS II

In Table 2–162, the column heads "VS" and "RT" represent AOS/VS and AOS/RT32, respectively. The entries in these columns have the following codes:

—	The operating system does not define a value for the given offset.
[no entry]	The operating systems handle the offset in the sameway.

Table 2–162. Contents of ?PROC Parameter Packet Extension for AOS/VS and AOS/RT32

Offset	VS	RT	Description
?PXSPI (dbl wd)			Packet identifier. Place 0 here.
?PXCPU		---	CPU number. Place here the ID of the CPU on which this process executes. AOS/RT32 does not use this value; supply -1.
?PXULOC		---	Place here the user locality as you have defined it. AOS/RT32 does not use this value; supply -1.
?PXLLOC		---	Legal user localities. This offset indicates the legal user localities that the new process may change to. The format is a bit map. If a bit is set, the new process may change to the corresponding user locality. Note that this format is the same one that ?PROFILE returns, so you can take the legal user localities from the profile and insert them directly into this extension packet. AOS/RT32 ignores this offset; supply -1.
?PXFLG		---	Extender flags word. This contains bit ?PFLL. If the bit is set, AOS/VS uses a default value for the son's legal locality. In other words, AOS/VS would NOT refer to offset ?PXLLOC to get the son's locality. If the bit is not set, AOS/VS obtains the son's locality from offset ?PXLLOC. AOS/RT32 does not use this information; supply -1.
?PXPGN (dbl wd)			Reserved. (Set to -1.)
?PXPUN (dbl wd)			Byte pointer to the ?PROC-related username that the new process will be created with, but only if the process has Superprocess privilege and has turned it on. Otherwise, supply -1. Normally only Data General software such as EXEC supplies a byte pointer here.
?PXUPID (8 words)			Reserved. (Set these 8 words to 0.)

Table 2–163. Contents of ?PROC Parameter Packet Extension for AOS/VS II

Offset	Description
?PXSPI	Packet identifier; ?PXSID defines the packet for AOS/VS II use.
?PXCPU	CPU number. Place the ID of the CPU on which this process executes.
?PXULOC	Specify the user locality.
?PXLLOC	Legal user localities. This offset indicates the legal user localities to which the new process may change. The format is a bit map. If a bit is set, the new process may change to the corresponding user locality. Note that this format is the same one that ?PROFILE returns, so you can take the legal user localities from the profile and insert them directly into this extension packet.
?PXFLG	Extender flags word. This word contains bit ?PFDLL. If the bit is set, AOS/VS uses a default value for the son's legal locality. In other words, AOS/VS II would NOT refer to offset ?PXLLOC to get the son's locality. If the bit is not set, AOS/VS II obtains the son's locality from offset ?PXLLOC.
?PXPGN	Reserved. (Set to 0.)
?PXPUN	Byte pointer to the ?PROC-related username that the new process creates, but only if the process has Superprocess privilege turned on. Otherwise, supply -1. Normally only Data General software such as EXEC supplies a byte pointer here.
?PXPAG	Byte pointer to the group access control list. A group access control list is a double, null-terminated list of group names. A group name is a null-terminated, ASCII string with a maximum length of 16 bytes, including the null character. A group name corresponds to a filename in the :GROUPS directory. If you specify -1, use the parent's group access control list. If you specify 0, use a null group access control list. If you call ?PROC for AOS/VS II with no extension packet or with the AOS/VS and AOS/RT32 extension packet, the call behaves as if you specified a -1 in ?PXPAG (process inherits its parent's list of groups).
?PXPGI	Length of the group access control list.
?PXRES ?PXRES0 ?PXRES1	Reserved. (Set to 0.) Reserved. (Set to 0.) Reserved. (Set to 0.)

Sample Packet

```
PKT:    .BLK    ?PLTH           ;Allocate enough space for packet.
                                ;Packet length = ?PLTH.

        .LOC    PKT+?PFLG       ;Process creation options.
        .WORD   ?PBRK+0         ;Create a break file if process traps
                                ;and son is a swappable process.

        .LOC    PKT+?PPRI       ;Priority of son process
        .WORD   -1              ;is same as caller's priority (-1 is
                                ;the default).

        .LOC    PKT+?PSNM       ;Byte pointer to pathname of program
                                ;file for new process to execute.
        .DWORD  PATH*2          ;Byte pointer to PATH.

        .LOC    PKT+?PIPC       ;Address of IPC message header.
        .DWORD  -1              ;No IPC message header (-1 is the
                                ;default).

        .LOC    PKT+?PNM        ;Byte pointer to new process's simple
                                ;process name.
        .DWORD  -1              ;Son's simple process name is ASCII
                                ;representation of its PID.

        .LOC    PKT+?PMEM       ;Maximum number of log pages in new
                                ;process.
        .DWORD  20.             ;Maximum of 20. log pages in new
                                ;process.

        .LOC    PKT+?PDIR       ;Byte pointer to name of son's
                                ;working directory.
        .DWORD  -1              ;Same initial working directory as
                                ;the caller's (-1 is the default).

        .LOC    PKT+?PCON       ;Byte pointer to name of new
                                ;process's @CONSOLE device.
        .DWORD  0               ;No device is associated with
                                ;@CONSOLE.

        .LOC    PKT+?PCAL       ;Maximum number of system calls son
                                ;can issue at the same time.
        .WORD   3.              ;Son can issue three system calls
                                ;at a time.

        .LOC    PKT+?PWSS       ;Maximum working set size for son.
        .WORD   -1              ;No limit to working set size for
                                ;son.  (The caller must have the
                                ;?PVWS privilege to set this
                                ;parameter.)
```

```
.LOC    PKT+?PUNM       ;Byte pointer to son's username.
.DWORD  -1              ;Same username as caller's (-1 is
                        ;the default).
.LOC    PKT+?PPRV       ;Son's file access privileges.
.WORD   -1              ;Son has all caller's file access
                        ;privileges.
.LOC    PKT+?PPCR       ;Maximum number of sons this son can
                        ;create.
.WORD   1               ;This son can create one son.
.LOC    PKT+?PWMI       ;Minimum working set size.
.WORD   -1              ;No working set minimum.
.LOC    PKT+?PIFP       ;Byte pointer to pathname of son's
                        ;@INPUT file.
.DWORD  -1              ;Same @INPUT file as caller.
.LOC    PKT+?POFP       ;Byte pointer to pathname of son's
                        ;@OUTPUT file.
.DWORD  -1              ;Same @OUTPUT file as caller.
.LOC    PKT+?PLFP       ;Byte pointer to pathname of son's
                        ;@LIST file.
.DWORD  0               ;There is no @LIST file.
.LOC    PKT+?PDFP       ;Byte pointer to pathname of son's
                        ;@DATA file.
.DWORD  0               ;There is no @DATA file.
.LOC    PKT+?SMCH       ;Maximum CPU time allotted to son.
.DWORD  -1              ;Son receives remainder of father's
                        ;time limit.
.LOC    PKT+?PLTH       ;End of packet.
```

Notes

- See the description of ?XPSTAT in this chapter, which returns the group access control list for a process.

AOS/VS

?PROFILE *[packet address]*
error return
normal return

Input

AC0	Reserved (Set to 0.)
AC1	Reserved (Set to 0.)
AC2	Address of the ?PROFILE packet, unless you specify the address as an argument to ?PROFILE

Output

AC0	Unchanged
AC1	Unchanged
AC2	Address of the ?PROFILE packet

Error Codes in AC0

ERBTL	Source field is too large
EREDX	Profile entry nonexistent
EREOF	End of file
ERIRB	Destination field too small
ERPAI	Profile access has already been initialized
ERPAN	Profile access has not been initialized
ERPFD	Error in profile field descriptor packet
ERPRE	Invalid parameter passed as a system call argument
ERPRV	Caller not privileged for this action
ERPVS	Packet revision not supported
ERRVN	Reserved value not zero
ERUNM	Illegal username
ERVBP	Invalid byte pointer passed as a system call argument
ERVWP	Invalid word pointer passed as a system call argument

Why Use It?

Use this system call to obtain information about, create, delete, and change profiles.

Who Can Use It?

You must have Superuser privilege to issue this call, but there are no restrictions concerning file access.

What It Does

This system call lets you perform all the functions of utility program PREDITOR (user profile editor) from within your application program. These functions include the following:

- Creating a profile.
- Renaming a profile.
- Deleting a profile.
- Initiating access to a profile.
- Terminating access to a profile.
- Reading a field in a profile.
- Updating a field in a profile.

You may perform only one of these functions each time you issue ?PROFILE. In order to read a field or update one or more fields, begin by initiating access (function code ?PFIAC in offset ?PFFC). Then, read or update as many fields as you wish. Finish by terminating access (function code ?PFTAC in offset ?PFFC).

Figure 2–182 shows the structure of the ?PROFILE parameter packet, and Table 2–164 describes its contents.

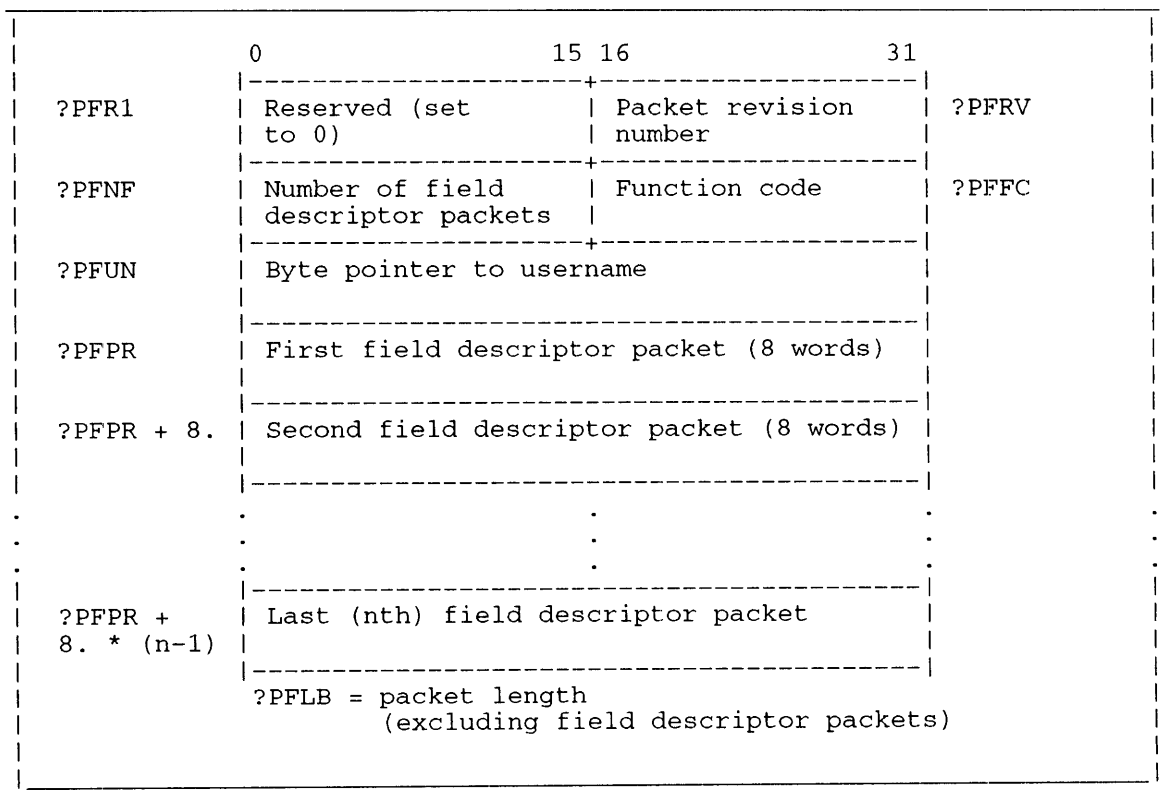


Figure 2–182. Structure of ?PROFILE Parameter Packet

Table 2–164. Contents of ?PROFILE Parameter Packet

	0 15	16 31	
?PFR1	Reserved (Set to 0.)	Current revision of this system call packet. Place ?PKR0 here.	?PFRV
?PFNF	Number of field description packets. They begin at offset ?PFPR. This number is zero for some functions (create a profile) and at least one for other functions (rename a profile).	Function code for the profile. ?PFCRE - Create ?PFREN - Rename ?PFDEL - Delete ?PFIAC - Initiate access ?PFTAC - Terminate access ?PFRDF - Read field ?PFUFD - Update field	?PFFC
?PFUN (doubleword)	Byte pointer to username. Null terminate this string.		
?PFPR (8. words)	Field descriptor packet number 1. See Figure 2-183.		
?PFPR + 8. (8. words)	Field descriptor packet number 2. See Figure 2-183.		
. . .	. . .		
?PFPR + 8. * (n-1)	Field descriptor packet number n. See Figure 2-183.		

Figure 2–183 shows the structure of each field descriptor packet. The packets (if any) begin at offset ?PFPR of the ?PROFILE parameter packet.

	0 15	16 31	
?PFFD	Field descriptor		
?PFDP	Data address		
?PFDL	Buffer length	Actual data length	?PFAL
?PFER	Error code	Reserved (set to 0)	?PFR3

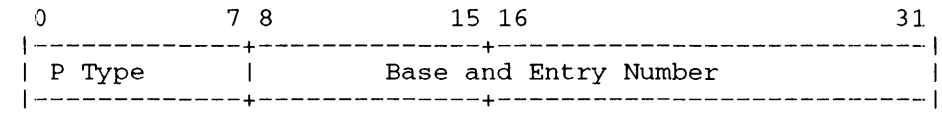
?PFLE = packet length

Figure 2–183. Structure of ?PROFILE Field Descriptor Packet

Here are the 8 words of each field descriptor packet. Their descriptions continue over the next several pages.

?PFFD —(doubleword) field descriptor.

It describes the data item in the profile that you want to retrieve or modify. Its format is



"P Type" means parameter type. Its values and their meanings are

?PFBY — Offsets ?PFDP and ?PFDL are byte oriented.
?PFBI — Offset ?PFDP contains a value; ignore ?PFDL.
?PFWD — Offsets ?PFDP and ?PFDL are word oriented.

The 8 bits of the base usually contain ?PFSE. See PARU.32 for other possible values of these bits.

The rightmost 16 bits of offset ?PFFD (entry number) have the following possible values. Entries with a bit length of 1 bit are set/returned in the most significant bit. (1S0). For example, suppose you want to know if a profile has the Superuser privilege. Set offset ?PFFD to ?PFBI!?PFSE!?PSUSER (or ?PFBI+?PFSE+?PSUSER) and the other 7 words of the field descriptor packet to zero. The operating system returns 1S0 in offset ?PFDP if Superuser privilege is present and 0S0 otherwise.

- ?PPASSWD — system password, 16 bytes including a null byte (if bit ?PNCRYPT is not set). If bit ?PNCRYPT is set, the password is already encrypted. To encrypt a password you must convert its letters to uppercase, pad it with bytes of <377> to a length of 16 bytes, and issue system call ?PWDCRYP. See the "What It Does" section in the description of ?PWDCRYP.

- ?PICROG — initial program, 64 bytes including a null byte.

- ?PICCFN — initial interprocessor communications (IPC) filename, 64 bytes including a null byte.

- ?PMXSONS — maximum number of sons, 1 word.

- ?PSOPIO — son's priority, 1 word.

- ?PDISKLM — user's CPD (control point directory) limit, 2 words.

- ?PLOGON — date/time of last logon, 6 words; internal format is

  word 0 — seconds
  word 1 — minutes
  word 2 — hours
  word 3 — days
  word 4 — months
  word 5 — years

?PFFD — (doubleword) field descriptor, continued.

- ?PBATCHP — batch priority, 1 word.
- ?PCNSPRV — terminal usage privilege, 1 bit.
- ?PBCHPRV — batch usage privilege, 1 bit.
- ?PMODPRV — modem usage privilege, 1 bit.
- ?PVCNPRV — virtual terminal usage privilege, 1 bit (negative logic).
- ?PRRAPRV — remote resource access privilege, 1 bit (negative logic).
- ?PPWDPRV — change password privilege, 1 bit (negative logic).
- ?PNCRYPT — system password encrypted indicator, 1 bit.
- ?PMGSYS — system manager privilege, 1 bit.
- ?PCOMMNT — user comment, 80 bytes including a null byte.
- ?PBWSS — batch working set size, 2 words; the internal format is

  word 0 — maximum
  word 1 — minimum

- ?PBLMEM — batch logical memory, 2 words; the internal format is

  word 0 — low order
  word 1 — high order

- ?PNBWSS — nonbatch working set size, 2 words; the internal format is

  word 0 — maximum
  word 1 — minimum

- ?PNBLMEM — nonbatch logical memory, 2 words; the internal format is

  word 0 — low order
  word 1 — high order

- ?PMYSONS — many sons privilege, 1 bit.
- ?PCHTYP — change type privilege, 1 bit.
- ?PCHPRI — change priority privilege, 1 bit.
- ?PPDPMGR — define PMGR privilege, 1 bit.
- ?PPRNBLK — PROC no-block privilege, 1 bit.
- ?PCHUSER — change username privilege, 1 bit.
- ?PACDEV — access devices privilege, 1 bit.
- ?PUIPCS — use IPC privilege, 1 bit.
- ?PSUSER — Superuser privilege, 1 bit.
- ?PPSUPP — Superprocess privilege, 1 bit.

?PFFD — (doubleword) field descriptor, continued.

- ?PWSON — wide son privilege, 1 bit.
- ?PMCTS — (read–only field) memory constraints, 10 words; the internal format is

words 0–1	?PBWSS
words 2–3	?PBLMEM
words 4–5	?PNBWSS
words 6–7	?PNBLMEM
word 8	?PSOPIO
word 9	?PBATCHP

- ?PHRDPRV — (read–only field) hard privileges, 1 word; the internal format is

bits 0–2	0
bit 3	?PCHWSSL
bit 4	?PWSON
bit 5	?PPSUPP
bit 6	?PMGSYS
bit 7	?PUIPCS
bit 8	?PACDEV
bit 9	?PCHUSER
bit 10	?PPRNBLK
bit 11	?PPDPMGR
bit 12	?PCHPRI
bit 13	?PSUSER
bit 14	?PCHTYP
bit 15	?PMYSONS

- ?PSFTPRV — (read–only field) soft privileges, 1 word; the internal format is

bits 0–9	0
bit 10	?PPWDPRV (negative logic)
bit 11	?PRRAPRV (negative logic)
bit 12	?PVCNPRV (negative logic)
bit 13	?PMODPRV
bit 14	?PBCHPRV
bit 15	?PCNSPRV

- ?PPRCINF — (read–only field) PROC information, 13 words; the internal format is

word 0	?PMXSONS
word 1	?PSFTPRV
word 2	?PHRDPRV
words 3–12	?PMCTS

- ?PINTDIR — initial directory, less than or equal to 64 bytes including a null byte.
- ?PCHWSSL — change WSS limit privilege, 1 bit.

?PFFD — (doubleword) field descriptor, continued.

- ?PDLOCY — default terminal (interactive) locality, 1 word.
- ?POLOCY — other terminal (interactive) localities the process may use, 1 word.
- ?PQLOCY — other terminal (interactive) localities enable indicator, 1 word.
- ?PRCRYPT — remote password encrypted flag, 1 bit.
- ?PRPSSWD — remote password, less than or equal to 16 bytes including a null byte.
- ?PFVER — system profile revision, 2 words.
- ?PN1FLG — flag word, 1 word.
- ?PN2FLG — flag word, 1 word.
- ?PRHRDPR — remote hard privileges, 1 word.
- ?PRSFTPR — remote soft privileges, 1 word.
- ?PDBLOCY — default batch (noninteractive) locality, 1 word.
- ?POBLOCY — other batch (noninteractive) localities the process may use, 1 word.
- ?PQBLOCY — other batch (noninteractive) localities enable indicator, 1 word.

Place ?PFBY in offset ?PFFD for the rename function.

?PFDP — data address (doubleword). Enter the byte or word address of the buffer that the operating system returns data in, the byte or word address of the data you want to write, or the data itself that you want to write. You specify the address type (byte or word) by the parameter type portion of offset ?PFFD. For the rename function, offset ?PFDP contains a byte pointer to the null–terminated new username.

?PFDL — buffer length. Enter the length of either the data buffer (for reads) or the length of the data you want to write (for updates). You specify the data size in the units declared by the parameter type portion of offset ?PFFD. Supply zero in ?PFDL for the rename function.

?PFAL — actual data length. After a read, the operating system returns the actual size of the specified field. After an update, the operating system returns the data length specified in offset ?PFDL. You specify the data size in the units declared by the parameter type portion of offset ?PFFD.

?PFER — error code. If the operating system detects an error when processing the field descriptor or when retrieving data that this field descriptor specifies, it returns ERPFD in AC0 and the actual error code in offset ?PFER.

?PFR3 — reserved; set to 0.

Notes

- See the description of ?PWDCRYP in this chapter.
- Creating a profile does not implicitly create an initial directory for the user. You must use the ?CREATE or ?XCREATE system call to create an initial directory.

?PRRDY
error return
normal return

Input		Output	
AC0	Priority number of the tasks to ready	AC0	Unchanged
AC1	Reserved (Set to 0.)	AC1	Undefined
AC2	Reserved (Set to 0.)	AC2	Undefined

Error Codes in AC0

No error codes are currently defined.

Why Use It?

You can use ?PRRDY to ready all suspended tasks of a given priority level to be rescheduled and executed. As we noted earlier, the operating system neither executes nor schedules suspended tasks. Given more than one ready task at the same priority, the operating system schedules and executes them on a round–robin basis. Round robin means that the first task in the priority queue executes first, and so on, down the list. Note that ?PRRDY does not change the relative positions of the tasks within a priority level.

Who Can Use It?

There are no special process privileges needed to issue this call, and there are no restrictions concerning file access.

What It Does

?PRRDY readies, that is, lifts from suspension, all tasks of the priority level that you specify in AC0, provided the tasks were suspended with ?SUS, ?IDSUS, or ?PRSUS. If the target tasks were suspended for other reasons, such as outstanding system calls, ?XMT, or ?REC, the operating system waits for the appropriate events to complete before it readies the tasks. If there are no tasks at the specified priority level, the operating system ignores the ?PRRDY and takes the normal return.

When you use ?PRRDY to ready tasks that have lower priorities than the calling task, the operating system reschedules the tasks.

Notes

- See the descriptions of ?SUS, ?IDSUS, ?PRSUS, ?XMT, and ?REC in this chapter.

?PRSUS
error return
normal return

Input

AC0	Priority level of the tasks to suspend
AC1	Reserved (Set to 0.)
AC2	Reserved (Set to 0.)

Output

AC0	Unchanged
AC1	Undefined
AC2	Undefined

Error Codes in AC0

No error codes are currently defined.

Why Use It?

?PRSUS allows you to suspend more than one task at a time. For example, you might group several tasks together under the same priority level, and then suspend the entire set with ?PRSUS.

Who Can Use It?

There are no special process privileges needed to issue this call, and there are no restrictions concerning file access.

What It Does

?PRSUS is the inverse of ?PRRDY; that is, it suspends all tasks of a given priority level. The tasks remain suspended until you ready them by issuing a ?PRRDY against the same priority level, an ?IDRDY against each task, or a ?PRKIL to terminate all tasks at that level.

Before you issue ?PRSUS, load AC0 with the priority level of the tasks to suspend. Note that ?PRSUS also suspends the calling task, if it is a member of the priority group specified in AC0. If there are no tasks at the specified priority level, the operating system ignores the ?PRSUS and takes the normal return.

Notes

- See the description of ?PRRDY in this chapter.

?PSTAT *[packet address]*
error return
normal return

Operating System Differences	
Accumulator	
Input and Output	None
Error Codes	None
Parameter Packet	Some

Input

		Output	
AC0	One of the following: • PID or VPID of the target process • Byte pointer to the name of the target process • –1 for status information about the calling process	AC0	Unchanged
AC1	One of the following: • –1 if AC0 is a byte pointer • Any other value if AC0 contains either –1 or a PID	AC1	Unchanged
AC2	Address of the ?PSTAT packet, unless you specify the address as an argument to ?PSTAT	AC2	Unchanged

Error Codes in AC0

ERIRB	Insufficient room in buffer (for username, ?PROC–related username, memory descriptors, or process group name)
ERPNM	Illegal process name
ERPRH	Attempt to access process not in hierarchy
ERPVS	Packet revision not supported
ERVBP	Illegal byte pointer in AC0
ERVWB	Packet failed read/write validation
ERVWP	Illegal word pointer in AC2

Why Use It?

Like ?RUNTM, ?WHIST, and ?XPSTAT, ?PSTAT helps you to determine how well a process competes with others of the same type for system resources, such as memory and CPU time. Note that the operating system updates most of the ?PSTAT information as the process executes.

Who Can Use It?

There are no special process privileges needed to issue this call, and there are no restrictions concerning file access.

What It Does

?PSTAT returns internal statistics about the calling process or about any process the caller specifies in AC0. The operating system returns the statistics in the packet. Figure 2–185 shows the structure and Table 2–165 describes the contents of the ?PSTAT packet.

The ?PSAL words in the packet return information about son PIDs from 1 through 255. However, system call ?SONS provides you with the information in these offsets — and with information about son PIDs greater than 255. System call ?XPSTAT has almost all the functionality of ?PSTAT along with much new functionality and considerably less execution time. We recommend that you use ?XPSTAT and/or ?SONS instead of ?PSTAT in new programs.

In addition to the process statistics, the operating system returns seven sets of memory descriptors in the packet, one descriptor for each of the rings 1 through 7. Each descriptor contains information about the program currently running in that ring. Figure 2–184 shows the memory descriptor structure.

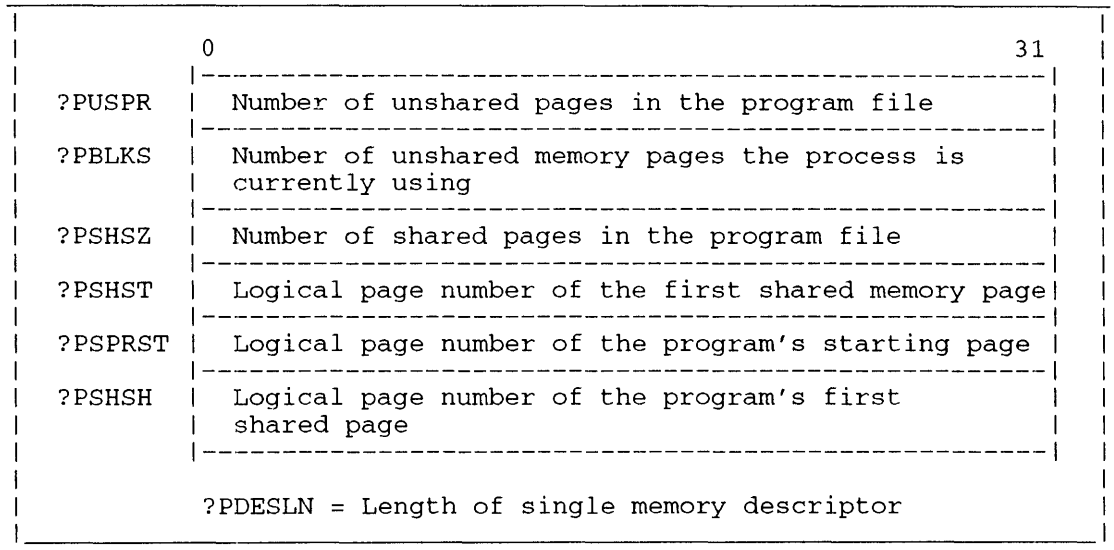
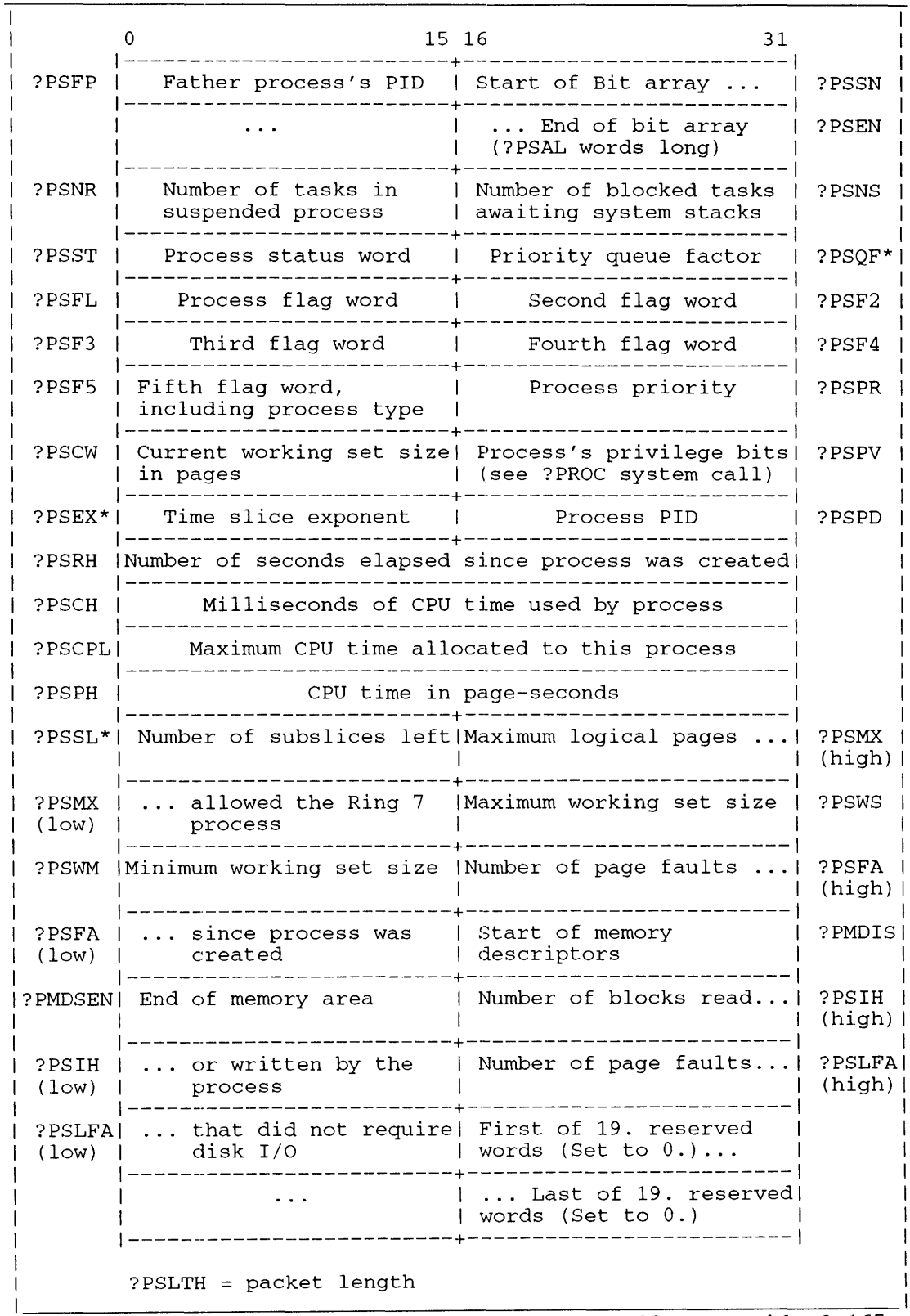


Figure 2–184. Structure of ?PSTAT Memory Descriptor



* This offset differs between AOS/VS and AOS/RT32. See Table 2-165.

Figure 2-185. Structure of ?PSTAT Packet

In Table 2–165, the column heads "VS" and "RT" represent "AOS/VS" and "AOS/RT32," respectively. The entries in these columns have the following codes:

——	The operating system does not define a value for the given offset.
U	The operating system handles the offset differently (uniquely).
[no entry]	The operating systems handle the offset in the sameway.

Table 2–165. Contents of ?PSTAT Parameter Packet

Offset	VS	RT	Description
?PSFP			PID of the process's parent process. The operating systems reserve a 16-bit word for the small PID. User programs that mask the value returned to 8 bits (length of "low pids") will not return correct PID values for creating processes with hybrid- or anyPIDs.
?PSSN			Bit array that identifies those processes subordinate to the target processes that have small PIDs. (Use the ?SONS system call for PID information on subordinate processes with hybrid- or anyPIDS.) The position of the bits indicates the PID numbers. If Bit 33 is set, for example, the target process has a subordinate process with a PID of 33. This bit array is ?PSAL words long. Ignore Bit 0.
?PSEN			End of the bit array.
?PSNR			Number of tasks in the process currently suspended on ?IREC system calls.
?PSNS			Number of tasks that are blocked awaiting system stacks.
?PSST			Process status word (Bit 6: process is blocked).
?PSQF		---	Priority queue factor (indicates the process's priority for execution relative to other processes of the same type). AOS/RT32 does not return this information.
?PSFL			Process flag word ?PSSP is set and ?PSPP is not set -- process is swappable ?PSSP is not set and ?PSPP is set -- process is pre-emptible ?PSSP is not set and ?PSPP is not set -- process is resident
?PSF2			Second flag word.

(continued)

Table 2–165. Contents of ?PSTAT Parameter Packet

Offset	VS	RT	Description
?PSF3			Third flag word.
?PSF4			Fourth flag word.
?PSF5			Fifth flag word. It returns the process type as follows: ?PSXPT is not set and ?PSHRP is not set -- type A process ?PSXPT is set and ?PSHRP is not set -- type B process ?PSXPT is set and ?PSHRP is set -- type C process
?PSPR			Process priority.
?PSCW			Current number of shared and unshared pages in process's logical address space (the OS user-defined "working set" size).
?PSPV			Process's privilege bits. (See ?PROC in this system call dictionary for information on privileges.)
?PSEX	U	U	Current time slice exponent (value indicating the amount of CPU time currently allocated to the target process). Systems determine value heuristically using "subslice," a constant, and "s", the time-slice exponent: AOS/VS: (time slice exponent range of 1 - 6) time slice = subslice * 2**s AOS/RT32: Returns 0. (See offset ?PSSL in this table and the functional description below.)
?PSPD			PID of the process.
?PSRH (dbl wd)			Number of seconds that have elapsed since the process was created.
?PSCH (dbl wd)			CPU time (in milliseconds) used by the process.
?PSCPL (dbl wd)			Maximum CPU time allocated to this process.
?PSPH (dbl wd)			CPU time (in page-seconds).

(continued)

Table 2–165. Contents of ?PSTAT Parameter Packet

Offset	VS	RT	Description
?PSSL	U	U	Number of remaining subslices in the process's time slice. This value is based on the current time slice exponent returned in ?PSEX. If the time slice exponent is 1, an AOS/VS process has two remaining subslices (that is, 2**1). AOS/RT32 does not break time slices into subslices, so it returns 0.
?PSMX (dbl wd)			Maximum number of logical pages allowed the process in Ring 7.
?PSWS			Maximum working set size in pages.
?PSWM			Minimum working set size in pages.
?PSFA (dbl wd)			Number of page faults since the process was created (with ?PROC).
?PMDIS			Start of memory descriptors.
?PMDSEN			End of memory area.
?PSIH (dbl wd)			Number of blocks read or written by the process.
?PSLFA (dbl wd)			Number of page faults that did not require disk input/output.

(concluded)

Notes

- See the descriptions of ?RUNTM and ?WHIST in this chapter.
- The parameters shown in Figure 2–184 are offsets from the memory descriptor base address. To get the base address for a particular ring, use the following formula:

  <base_of_packet> + ?PMDIS + (?PDESLN * (ring number–1))

  where

  ?PMDIS — is the starting offset in the ?PSTAT packet that contains the memory descriptors.

  ?PDESLN — is the length of each memory descriptor.

  Assume, for example, that you loaded a program file into Ring 5 with the ?RINGLD system call. To obtain the base address of the descriptor for that ring, you would use the following formula:

  <base_of_packet> + ?PMDIS + (?PDESLN * 4)

?PSTAT Continued

Sample Packet

```
PKT: .BLK    ?PSLTH           ;Allocate enough space for packet.
                              ;Packet length = ?PSLTH.
     .LOC    PKT+?PSFP        ;PID of father process.
     .WORD   0
     .LOC    PKT+?PSSN        ;Bit array showing PIDs of process's
     .WORD   0                ;sons.
     .LOC    PKT+?PSEN        ;End of bit array.
     .WORD   0
     .LOC    PKT+?PSNR        ;Number of tasks in the process that
     .LOC    PKT+?PSSN+?PSAL  ;would also work.
     .WORD   0                ;are currently suspended on ?IREC
                              ;system calls.
     .LOC    PKT+?PSNS        ;Number of tasks that are blocked
     .WORD   0                ;awaiting system stacks.
     .LOC    PKT+?PSST        ;Process status word.
     .WORD   0
     .LOC    PKT+?PSQF        ;Priority queue factor.
     .WORD   0
     .LOC    PKT+?PSFL        ;Process flag word.
     .WORD   0
     .LOC    PKT+?PSF2        ;Second flag word.
     .WORD   0
     .LOC    PKT+?PSF3        ;Third flag word.
     .WORD   0
     .LOC    PKT+?PSF4        ;Fourth flag word.
     .WORD   0
     .LOC    PKT+?PSF5        ;Fifth flag word. (Reserved; set
     .WORD   0                ;to 0.)
     .LOC    PKT+?PSPR        ;Process priority.
     .WORD   0
     .LOC    PKT+?PSCW        ;Number of pages in current working
     .WORD   0                ;set.
     .LOC    PKT+?PSPV        ;Process's privilege bits.
     .WORD   0
     .LOC    PKT+?PSEX        ;Current time slice exponent for the
     .WORD   0                ;target process.
     .LOC    PKT+?PSPD        ;PID of the process.
     .WORD   0
     .LOC    PKT+?PSRH        ;Number of seconds elapsed since
     .DWORD  0                ;process creation.
     .LOC    PKT+?PSCH        ;CPU time process used (in
     .DWORD  0                ;milliseconds).
     .LOC    PKT+?PSCPL       ;Maximum CPU time allocated to this
     .DWORD  0                ;process.
     .LOC    PKT+?PSPH        ;Page usage over CPU time (in pages/
     .DWORD  0                ;second).
     .LOC    PKT+?PSSL        ;Subslices remaining in process's
     .WORD   0                ;time slice.
     .LOC    PKT+?PSMX        ;Maximum number of logical pages
     .DWORD  0                ;allowed this process in Ring 7.
     .LOC    PKT+?PSWS        ;Maximum working set size (in pages).
     .WORD   0
     .LOC    PKT+?PSWM        ;Minimum working set size (in pages).
     .WORD   0
     .LOC    PKT+?PSFA        ;Number of physical page faults since
     .DWORD  0                ;process creation.
     .LOC    PKT+?PMDIS       ;Start of memory descriptors.
     .WORD   0
     .LOC    PKT+?PMDSEN      ;End of memory area.
     .WORD   0
     .LOC    PKT+?PSIH        ;Number of blocks read or written by
     .DWORD  0                ;process.
     .LOC    PKT+?PSLFA       ;Number of logical page faults.
     .DWORD  0
     .LOC    PKT+?PSLTH       ;End of packet.
```

?PTRDEVICE — Controls input from a pointer device.

AOS/VS

?PTRDEVICE *[packet address]*
error return
normal return

Input

AC0	Reserved (Set to 0.)
AC1	Reserved (Set to 0.)
AC2	Address of the ?PTRDEVICE packet, unless you specify the address as an argument to ?PTRDEVICE.

Output

AC0	Unchanged or error code
AC1	Unchanged
AC2	Address of the ?PTRDEVICE packet

Error Codes Returned in AC0

ERIPP	Invalid pointer device parameter
ERPKT	Invalid packet ID
ERRVN	A reserved value not 0
ERVBP	Invalid byte pointer passed as a system call argument
ERVWP	Invalid address passed as a system call argument
ERWNE	The window you specified does not exist

Error codes from the file system

Why Use It?

?PTRDEVICE lets you control the type of input you receive from a pointer device. Input from a pointer device can tell you what the user is currently doing with the pointer device (such as moving the pointer into a window or pressing a pointer device button). You can also use ?PTRDEVICE to find out the current status and location of the pointer.

?PTRDEVICE is useful only if your program will run in an AOS/VS windowing environment. The call applies to a single window at a time. For example, if you want to find out the status of the pointer for every window belonging to your program, you must issue the ?PTRDEVICE call repeatedly, specifying a different window each time.

Who Can Use It?

There are no special process privileges needed to issue this call, and there are no restrictions concerning file access.

What It Does

?PTRDEVICE is a multifunctional system call. To perform a particular function, you set the offset ?PTRDEV_PKT.FUNC (in the main ?PTRDEVICE packet) to the function code you want. Table 2–166 lists the valid function codes and the functions they perform.

Table 2–166. ?PTRDEVICE Function Codes

Function Code	What It Lets You Do
?PTRDEV_SET_EVENTS	Specify the types of pointer events you want to receive in a window's input buffer. (See the section "Selecting the Pointer Events You Want.")
?PTRDEV_SET_DELTA	Specify how far the pointer device should move in a window before you receive a movement event. (See the section "Specifying a Pointer Delta.")
?PTRDEV_LAST_EVENT	Return information about the most recent pointer event to occur in a window. (See the section "Getting Information About the Last Pointer Event.")
?PTRDEV_SET_POINTER	Control the operation of the pointer within a window (enable or disable the pointer, set its shape, and/or control digitizing). (See the section "Controlling the Operation of the Pointer.")
?PTRDEV_GET_PTR_STATUS	Return the current pointer device settings for a window (which events you have requested, the current shape of the pointer, whether or not it is visible, and so on.) (See the section "Getting the Pointer Status.")
?PTRDEV_GENERATE_EVENT	Generate a pointer event as if it had come from the physical pointer device. Useful for moving the pointer or simulating pointer device input. (See the section "Moving the Pointer.")
?PTRDEV_GET_PTR_LOCATION	Return the current location of the pointer, and show which pointer device button(s) are currently down. (See the section "Getting the Location and Status of the Pointer.")
?PTRDEV_GET_TABLET_LOCATION	Return the current location of a tablet's puck or pen and its button state. (See the section "Getting the Tablet Status.")

The Main ?PTRDEVICE Packet

When you issue a ?PTRDEVICE call, you set up both a main packet (common to all functions of the call) and a subpacket (unique for each function). The structure for the main packet appears in Figure 2–186; we cover the subpackets later, when we discuss each function.

The ?PTRDEVICE call applies to one window at a time; in the main packet, you specify which window you want. You can specify a window in one of three ways:

- Channel number — When you open an I/O channel to a window (using ?OPEN), the operating system returns a channel number.
- Window pathname — When you create a window, you give it a window name. A window's pathname takes the form @PMAPn:windowname, where @PMAPn is the name of the pixel–mapped terminal that contains the window. The window pathname must end in a null character, <0>; it can be a maximum of ?MXPL characters long (including the null terminator).
- Window ID number — When you create a window, the operating system returns the window ID number in the main ?WINDOW packet.

Set the offset ?PTRDEV_PKT.FLAGS to indicate which method you are using to specify a target window (see Table 2–167). You can use only one method at a time. Fill in the appropriate field in the packet (channel number, pathname, or ID number), and set the other two fields to zero.

In addition to specifying the target window, you supply the following information in the main ?PTRDEVICE packet:

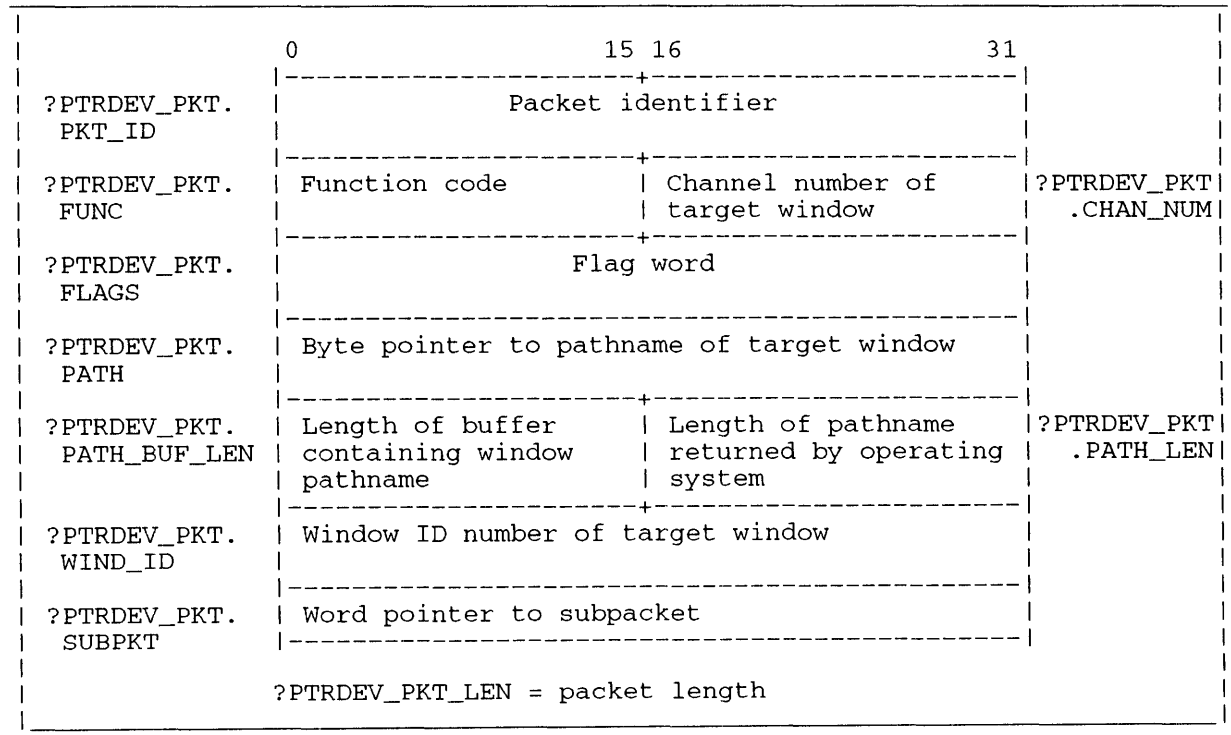


Figure 2–186. Structure of the ?PTRDEVICE Main Packet

Table 2–167. Contents of the ?PTRDEVICE Main Packet

Offset	Contents
?PTRDEV_PKT.PKT_ID (doubleword)	Packet identifier; set to ?PTRDEV_PKT_PKTID.
?PTRDEV_PKT.FUNC	The code for the function you want. You must always supply a function code. (Function codes are listed in Table 2-166.)
?PTRDEV_PKT.CHAN_NUM	Channel number of target window. When specifying the window by its channel number, you must also set flag ?PTRDEV_PKT.FLAGS.IN_CHAN in flag word ?PTRDEV_PKT.FLAGS. Set this offset to 0 when specifying the target window by its pathname or window ID.
?PTRDEV_PKT.FLAGS (doubleword)	Flag word ?PTRDEV_PKT.FLAGS.IN_PATH -- Set this flag when specifying the target window by its pathname. ?PTRDEV_PKT.FLAGS.IN_WIND_ID -- Set this flag when specifying the target window by its window ID. ?PTRDEV_PKT.FLAGS.IN_CHAN -- Set this flag when specifying the target window by its channel number.
?PTRDEV_PKT.PATH (doubleword)	Byte pointer to pathname of target window. When specifying the window by its pathname, you must also set flag ?PTRDEV_PKT.FLAGS.IN_PATH in flag word ?PTRDEV_PKT.FLAGS. Set this offset to 0 when specifying the target window by its channel number or window ID.

(continued)

Table 2–167. Contents of the ?PTRDEVICE Main Packet

Offset	Contents
?PTRDEV_PKT.PATH_BUF_LEN	Length of the buffer containing the window pathname. The buffer length must include the null terminator; the maximum buffer length is ?MXPL.
?PTRDEV_PKT.PATH_LEN	Set to 0. For functions that return a pathname, the length of the pathname is returned here.
?PTRDEV_PKT.WIND_ID (doubleword)	Window ID number of target window. When specifying the window by its window ID, you must also set flag ?PTRDEV_PKT.FLAGS.IN_WIND_ID in flag word ?PTRDEV_PKT.FLAGS. Set this offset to 0 when specifying the target window by its pathname or channel number.
?PTRDEV_PKT.SUBPKT (doubleword)	Word pointer to the subpacket for the function. If the function does not require a subpacket, set this offset to 0.

(concluded)

Using a Digitizing Tablet

Three ?PTRDEVICE functions return location values. These functions are

- ?PTRDEV_LAST_EVENT
- ?PTRDEV_GET_PTR_LOCATION
- ?PTRDEV_GET_TABLET_LOCATION

Digitize Option

When you use a digitizing tablet the word "location" depends on tablet options as shown in Figure 2–187. Before AOS/VS Revision 7.60 *location* always meant the location of the cursor relative to the origin of the active window. This definition still holds for all events that occur on the scaled portion of the tablet unless you select the *digitize* option that was first available in AOS/VS Revision 7.60. (You make this selection by placing ?PTRDEV_PTR_STATE_DIGITIZE in offset ?PTRDEV_SET_PTR.STATE for main function ?PTRDEV_SET_POINTER; see Table 2–172.) With digitizing enabled, ?PTRDEVICE returns the absolute location of the puck or pen on the tablet — as opposed to the location of the cursor on the screen. The resolution of the returned absolute location corresponds to the resolution of the tablet (2794 by 2794) to provide digitized applications with the best possible resolution.

The operating system converts the origin of the tablet to the upper–left corner for all options. (The actual origin of the hardware is the lower–left corner.) Depending on the options you choose, an application program might be interested in an absolute position over the entire tablet or else in just being able to address the lower portion of the tablet. In both cases you consider the absolute origin to be the upper–left corner of the tablet; furthermore, both the x and y coordinates range from 0 to 2793.

Selecting the digitize option results in the operating system's returning your requested tablet events in absolute coordinates across the entire tablet. Not selecting the digitize option results in the operating system's returning your requested tablet events in absolute coordinates only in the *dead space* of the tablet. See Figure 2–187. With the origin in the upper–left corner the dead space has absolute coordinates for x between 0 and 2793, and for y between 2236 and 2793.

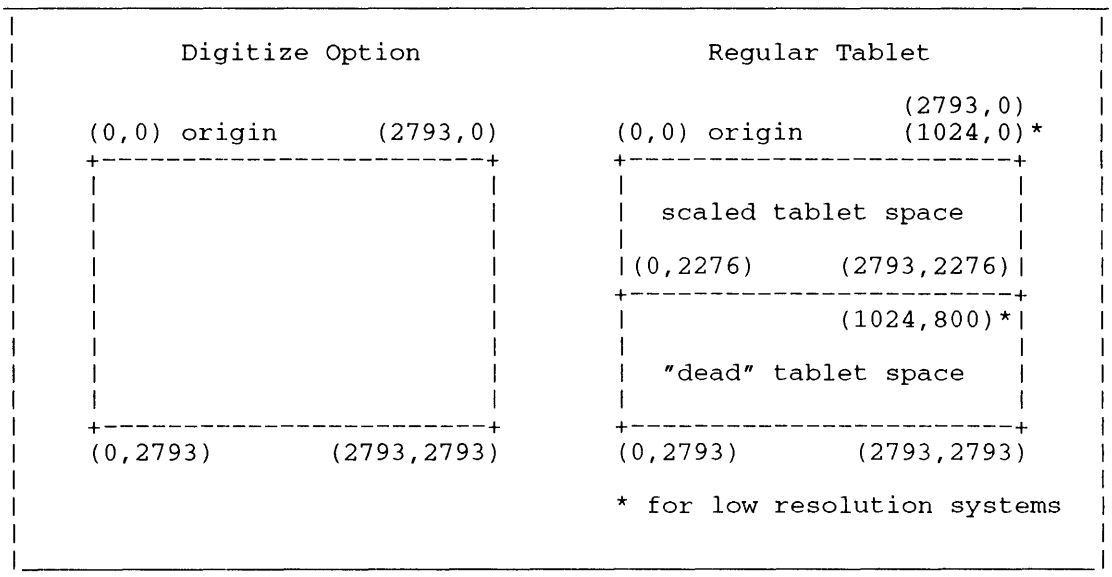


Figure 2–187. Tablet States

The *hotspot* of a cursor (the single pixel of a cursor image that defines the cursor location) cannot normally go off the physical screen. However, the operating system allows the cursor to completely leave the physical screen when tablet events occur; this way, the user is not confused when the cursor appears to be on the screen while events occur in the lower tablet space.

?PTRDEVICE does not support its ?PTRDEV_SET_DELTA function when you have enabled digitizing. This means that no filtering occurs and that the operating system returns movement events as the tablet reports them.

Tablet Location Example

Suppose you want to place a template over the *dead* area of the tablet. Your application will listen for events occurring in this area and respond accordingly. And, your application uses the scaled portion of the tablet as it always has (i.e., before AOS/VS Revision 7.60). See Figure 2–188.

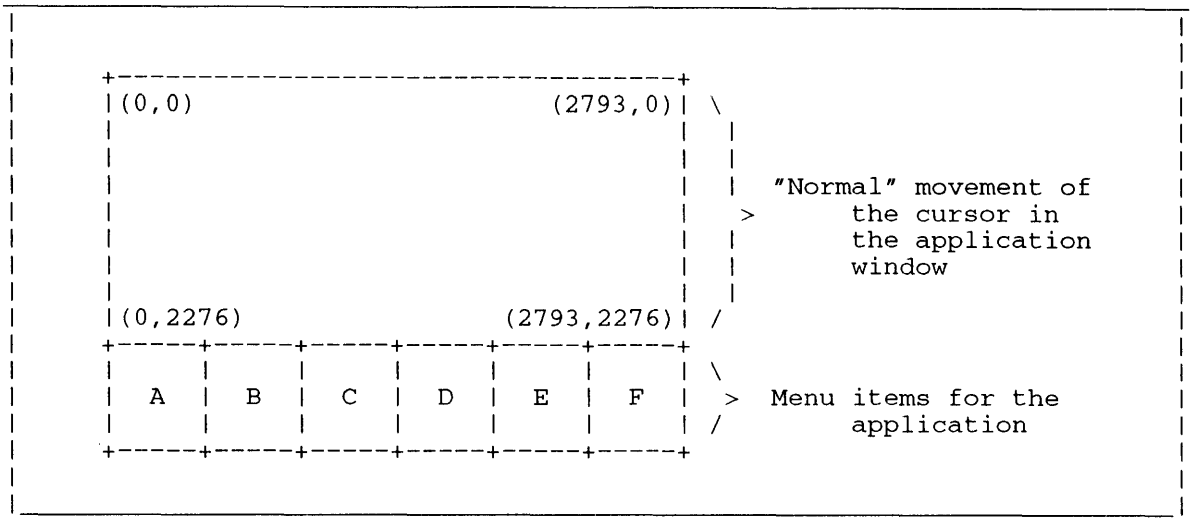


Figure 2–188. An Example of a Tablet Area Menu

Selecting the Pointer Events You Want

A pointer event is a 16–byte sequence that describes an action the user has just taken with the pointer device. The operating system inserts pointer events into the input data stream of the window in which the event took place. From the input stream, the events go into the window's input buffer, where they stay until you issue a ?READ on that window.

You must tell the operating system what types of pointer events you want it to send to each window. By default, each window receives no pointer events; to receive events, issue the ?PTRDEVICE call with function code ?PTRDEV_SET_EVENTS. In the main ?PTRDEVICE packet, specify the window for which you are setting the events, and provide a word pointer to the ?PTRDEV_SET_EVENTS subpacket. Figure 2–189 shows the subpacket structure.

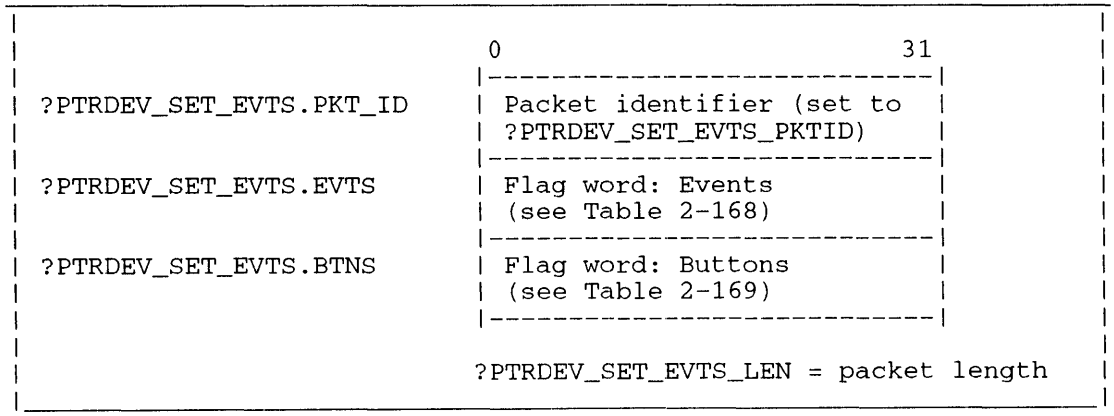


Figure 2–189. Structure of the ?PTRDEV_SET_EVENTS Subpacket

The ?PTRDEV_SET_EVENTS subpacket contains flag words that let you specify

- The pointer events you want to receive. You can specify one or more pointer events. (See Table 2–168.)
- The buttons in which you are interested (if you request pointer events that concern buttons). (See Table 2–169.)

Four tablet events became available with AOS/VS Revision 7.60. They allow your application to choose button events that occur in the dead portion of the tablet to return in absolute coordinates. Their function codes are

- ?PTRDEV_SET_EVTS.EVTS.TAB_MOVEMENT
- ?PTRDEV_SET_EVTS.EVTS.TAB_BTN_DOWN
- ?PTRDEV_SET_EVTS.EVTS.TAB_BTN_UP
- ?PTRDEV_SET_EVTS.EVTS.TAB_DBL_CLICK

When you choose one of these four tablet events and the operating system returns a tablet event, the returned location will have its origin in the upper–left corner of the tablet along with the resolution of the tablet device (2794 by 2794). The extent of the tablet's dead area is (0,2236) through (2793,2793). Any events occurring in the scaled portion of the tablet (corresponding to the screen) work the same in all revisions of the operating system: the location is relative to the origin of the active window and the resolution is that of the monitor device.

Also, when you choose one of these tablet events and there is more than one window in the active group, only the upfront window in the active group can receive tablet events. Other windows in the active group continue to receive other requested pointer events.

If you enable the digitize option, the operating system considers the location of all events to be absolute. Only the four special tablet events in the previous bulleted list are returned to your application when you have enabled digitizing; all other specified events are ignored. The origin is the upper–left corner of the tablet, and the x and y coordinates range from 0 to 2793.

Furthermore, the operating system returns error ERIPP (invalid pointer device parameter) if you specify one of the four special tablet events for a relative device such as a mouse.

Table 2–168 shows the pointer device events you can select by placing event flags in offset ?PTRDEV_SET_EVTS.EVTS. The actions of these events differ depending on whether or not you have enabled digitizing.

If you have enabled digitizing, the operating system returns the last four events (whose flag names contain TAB) over the entire tablet in absolute coordinates. Although the operating system does not return the other events with digitizing enabled, setting them does not cause an error. Why not? You might want to select the events after you have disabled digitizing. And, applications receive tablet events from the entire tablet only if you have enabled digitizing for the upfront window of the active group.

Table 2–168. Flags for Selecting Pointer Events (Flag Word ?PTRDEV_SET_EVTS.EVTS)

Flag	Description
?PTRDEV_SET_EVTS.EVTS.MOVEMENT	Set this flag to receive movement events that occur in this window. (A movement event occurs if the pointer moves while it is within the window.)
?PTRDEV_SET_EVTS.EVTS.WINDOW	Set this flag to receive window events that occur in this window. There are four kinds of window events: Enter window Exit window Activate window Deactivate window
?PTRDEV_SET_EVTS.EVTS.BTN_DOWN	Set this flag to receive button-down events for this window. (A button-down event occurs if the user presses a button while the pointer is within the window.)
?PTRDEV_SET_EVTS.EVTS.BTN_UP	Set this flag to receive button-up events for this window. (A button-up event occurs if the user releases a button while the pointer is within the window.)
?PTRDEV_SET_EVTS.EVTS.DBL_CLICK	Set this flag to receive double-click events for this window. (A double-click event occurs when the user presses and releases a button twice quickly.)
?PTRDEV_SET_EVTS.EVTS.TAB_MOVEMENT	Set this flag to receive tablet movement events in absolute coordinates.
?PTRDEV_SET_EVTS.EVTS.TAB_BTN_DOWN	Set this flag to receive tablet button-down events in absolute coordinates.
?PTRDEV_SET_EVTS.EVTS.TAB_BTN_UP	Set this flag to receive tablet button-up events in absolute coordinates.
?PTRDEV_SET_EVTS.EVTS.TAB_DBL_CLICK	Set this flag to receive tablet double-click events in absolute coordinates.

Table 2–169. Flags for Selecting Buttons (Flag Word ?PTRDEV_SET_EVTS.BTNS)

Flag	Description
?PTRDEV_SET_EVTS.BTNS.ONE	Set this flag to receive button events that concern pointer device button 1.
?PTRDEV_SET_EVTS.BTNS.TWO	Set this flag to receive button events that concern pointer device button 2.
?PTRDEV_SET_EVTS.BTNS.THREE	Set this flag to receive button events that concern pointer device button 3.

Specifying a Pointer Delta

The pointer delta is the number of pixels the pointer must move before the operating system sends you a movement event. To specify the pointer delta, issue the ?PTRDEVICE call, function code ?PTRDEV_SET_DELTA.

In the main ?PTRDEVICE packet, specify the window for which you are setting the pointer delta, and provide a word pointer to the ?PTRDEV_SET_DELTA subpacket.

In the ?PTRDEV_SET_DELTA subpacket, you specify the number of pixels the pointer must move in order to generate a movement event. You can specify different values for movement along the x– and y–axes. Figure 2–190 shows the subpacket structure; Table 2–170 details its contents.

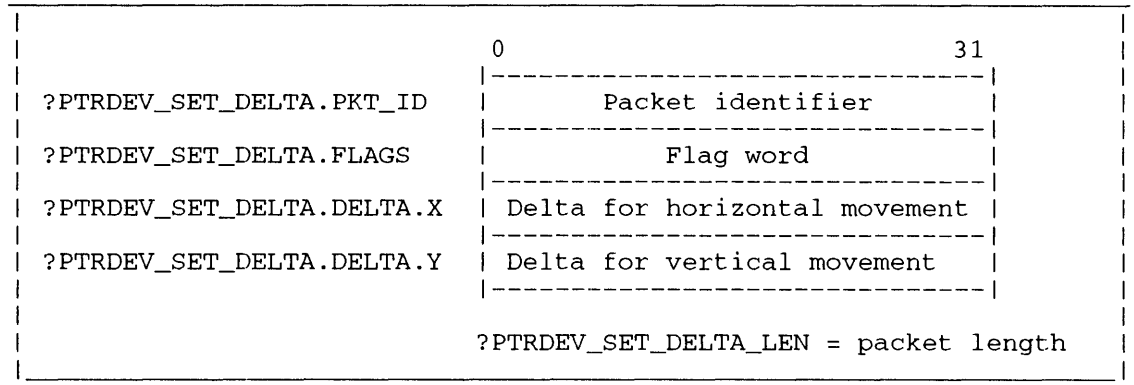


Figure 2–190. Structure of the ?PTRDEV_SET_DELTA Subpacket

Table 2–170. Contents of the ?PTRDEV_SET_DELTA Subpacket

Offset	Contents
?PTRDEV_SET_DELTA.PKT_ID (doubleword)	Packet identifier; set to ?PTRDEV_SET_DELTA_PKTID.
?PTRDEV_SET_DELTA.FLAGS (doubleword)	Flag word. If you are supplying a value, you must set the corresponding flag. ?PTRDEV_SET_DELTA.FLAGS.X -- Set this flag if you are specifying a delta for horizontal movement; set offset ?PTRDEV_SET_DELTA.DELTA.X to the delta value you want. ?PTRDEV_SET_DELTA.FLAGS.Y -- Set this flag if you are specifying a delta for vertical movement; set offset ?PTRDEV_SET_DELTA.DELTA.Y to the delta value you want.
?PTRDEV_SET_DELTA.DELTA.X (doubleword)	Number of pixels the pointer must travel horizontally in order to cause a movement event. This value must be an unsigned 32-bit value greater than 0. (You must also set the flag ?PTRDEV_SET_DELTA.FLAGS.X in the flag word ?PTRDEV_SET_DELTA.FLAGS.)
?PTRDEV_SET_DELTA.DELTA.Y (doubleword)	Number of pixels the pointer will travel vertically in order to cause a movement event. This value must be an unsigned 32-bit value greater than 0. (You must also set the flag ?PTRDEV_SET_DELTA.FLAGS.Y in the flag word ?PTRDEV_SET_DELTA.FLAGS.)

Getting Information About the Last Pointer Event

?PTRDEV_LAST_EVENT returns information about the last pointer event that was generated for a particular window.

In the main ?PTRDEVICE packet, specify the window for which you want the last event, and provide a word pointer to the ?PTRDEV_LAST_EVENT subpacket. The operating system returns information in the subpacket (see Figure 2–191 and Table 2–171).

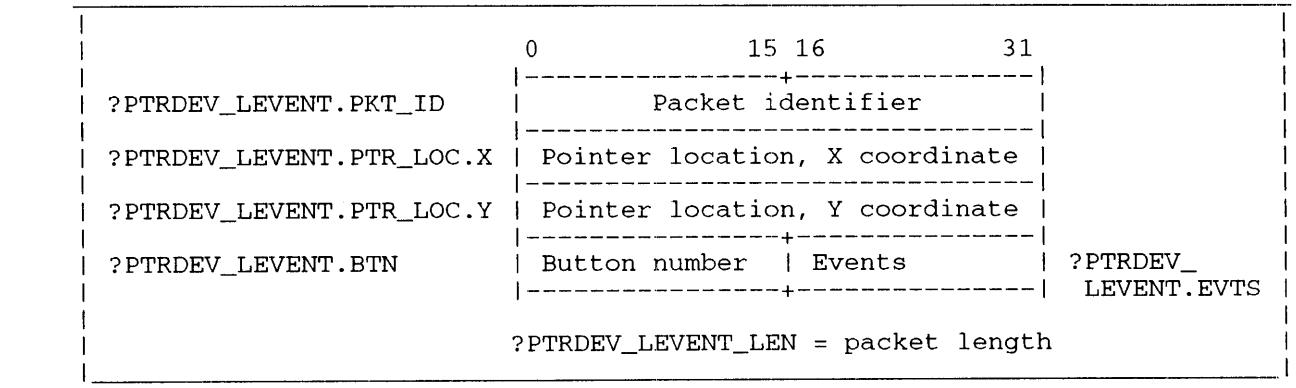


Figure 2–191. Structure of the ?PTRDEV_LAST_EVENT Subpacket

Table 2–171. Contents of the ?PTRDEV_LAST_EVENT Subpacket

Offset	Contents
?PTRDEV_LEVENT.PKT_ID (doubleword)	Packet identifier; set to ?PTRDEV_LEVENT_PKTID.
?PTRDEV_LEVENT.PTR_LOC.X (doubleword)	X coordinate of pointer location at the time the last event occurred, in pixels. This is either relative to the origin of a window's virtual terminal or else is an absolute tablet coordinate. (Returned by the OS.)
?PTRDEV_LEVENT.PTR_LOC.Y (doubleword)	Y coordinate of pointer location at the time the last event occurred, in pixels. This is either relative to the origin of a window's virtual terminal or else is an absolute tablet coordinate. (Returned by the OS.)
?PTRDEV_LEVENT.BTN	The number of the button involved in the event (if any). If the last event did not involve a button, this offset is set to -1. (Returned by the operating system.)
?PTRDEV_LEVENT.EVTS	The type of pointer event that last occurred (returned by the operating system). If you have selected both pointer and tablet events in the ?PTRDEV_SET_EVENTS subpacket in Figure 2-189, the operating system may return either type of event. If no event has yet occurred, this offset is set to -1. Event types are ?PTRDEV_EVENTS_MOVEMENT Movement ?PTRDEV_EVENTS_BTN_DOWN Button down ?PTRDEV_EVENTS_BTN_UP Button up ?PTRDEV_EVENTS_ENTER_WINDOW Enter window ?PTRDEV_EVENTS_EXIT_WINDOW Exit window ?PTRDEV_EVENTS_ACTIVATION Activation ?PTRDEV_EVENTS_DEACTIVATION Deactivation ?PTRDEV_EVENTS_DBL_CLICK Double click ?PTRDEV_EVENTS_TAB_MOVEMENT Tablet movement events in absolute coordinates ?PTRDEV_EVENTS_TAB_BTN_DOWN Tablet button-down events in absolute coordinates ?PTRDEV_EVENTS_TAB_BTN_UP Tablet button-up events in absolute coordinates ?PTRDEV_EVENTS_TAB_DBL_CLICK Tablet double-click events in absolute coordinates

Controlling the Operation of the Pointer

You use ?PTRDEVICE, function code ?PTRDEV_SET_POINTER, to control how the pointer looks when it is within a particular window or to control digitizing. You can

- Select a shape from the predefined pointer shapes.
- Make the pointer visible or invisible.
- Enable or disable digitizing.

In the main ?PTRDEVICE packet, specify the window for which you want to set the pointer, and provide a word pointer to the ?PTRDEV_SET_POINTER subpacket. Any changes you make to the pointer will be in effect only when the pointer is within the window you specify.

By default you create a window with the graphics pointer enabled and digitizing disabled. Enabling digitizing automatically disables the graphics cursor. Selecting either of the flags ?PTRDEV_PTR_STATE_ENABLE or ?PTRDEV_PTR_STATE_DISABLE in offset ?PTRDEV_SET_PTR.STATE will both disable digitizing and enable the graphics pointer.

Figure 2–192 shows the structure of the ?PTRDEV_SET_POINTER subpacket; Table 2–172 details its contents.

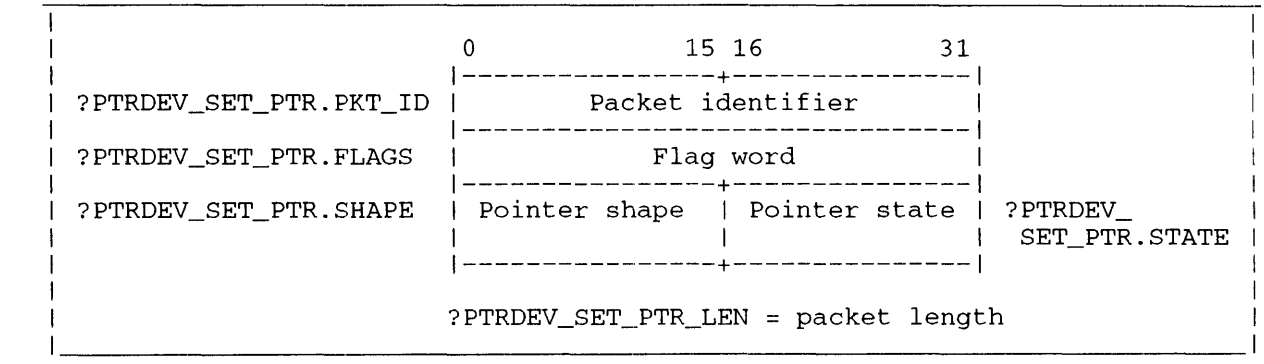


Figure 2–192. Subpacket for ?PTRDEV_SET_POINTER Subpacket

Table 2–172. Contents of the ?PTRDEV_SET_POINTER Subpacket

Offset	Contents
?PTRDEV_SET_PTR.PKT_ID (doubleword)	Packet identifier; set to ?PTRDEV_SET_PTR_PKTID.
?PTRDEV_SET_PTR.FLAGS (doubleword)	Flag word. Indicates which pointer attributes you are changing. ?PTRDEV_SET_PTR.FLAGS.SHAPE -- Set this flag if you are changing the pointer shape. ?PTRDEV_SET_PTR.FLAGS.STATE -- Set this flag if you are changing the state of the pointer (making it visible or invisible, or enabling digitizing).
?PTRDEV_SET_PTR.SHAPE	The pointer shape you want; available ones are ?PTRDEV_PTR_SHAPE_DEFAULT The operating system determines shape. ?PTRDEV_PTR_SHAPE_ARROW An arrow. ?PTRDEV_PTR_SHAPE_FINGER A pointing finger. ?PTRDEV_PTR_SHAPE_SXHAIR A small cross-hair. ?PTRDEV_PTR_SHAPE_FXHAIR A full-screen cross-hair. ?PTRDEV_PTR_SHAPE_HOURGLASS An hourglass.
?PTRDEV_SET_PTR.STATE	The state of the pointer (visible or invisible, and digitizing). By default, the pointer is visible. ?PTRDEV_PTR_STATE_ENABLE Set this flag to make the pointer visible and to disable digitizing. The OS returns pointer events for the scaled area of the tablet. ?PTRDEV_PTR_STATE_DISABLE Set this flag to make the pointer invisible and to disable digitizing. The OS returns pointer events for the scaled area of the tablet. ?PTRDEV_PTR_STATE_DIGITIZE Set this flag to enable digitizing for the target window. If it is the upfront window, then the graphics pointer does not appear on the entire screen and the OS does not return pointer events for the scaled portion of the tablet. The OS treats any event generated over the ENTIRE tablet as an absolute tablet event. If the target window is not the upfront window, then it receives no pointer events and the graphics pointer is unaffected. Select this state only for a digitizing tablet; otherwise, you receive error ERIPP.

Getting the Pointer Status

You can use ?PTRDEVICE, function code ?PTRDEV_GET_PTR_STATUS, to get the following information about the current state of the pointer:

- The current delta values.
- The events you have requested.
- The buttons you have requested.
- The current pointer shape.
- The current state of the pointer: enabled (visible) or disabled (invisible) or digitizing.

In the main ?PTRDEVICE packet, specify the window for which you want the pointer status, and provide a word pointer to the ?PTRDEV_GET_PTR_STATUS subpacket. The operating system returns information in the subpacket. Figure 2–193 shows the subpacket structures; Table 2–173 details its contents.

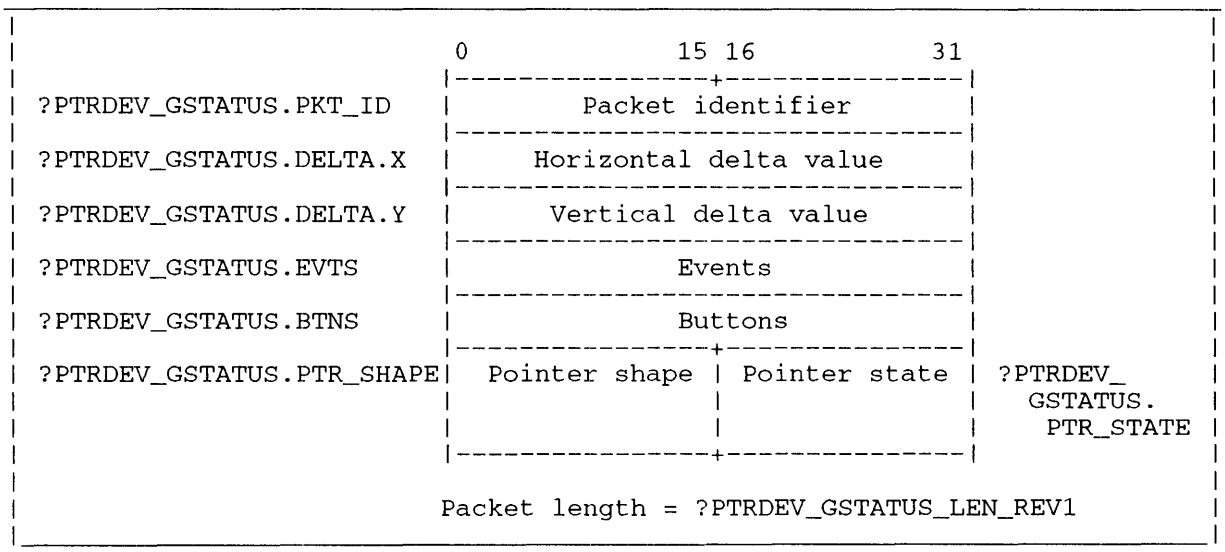


Figure 2–193. Structure of the ?PTRDEV_GET_PTR_STATUS Subpacket

Table 2-173. Contents of the ?PTRDEV_GET_PTR_STATUS Subpacket

Offset	Contents		
?PTRDEV_GSTATUS.PKT_ID (doubleword)	Packet identifier. Set to ?PTRDEV_GSTATUS_PKTID_REV1.		
?PTRDEV_GSTATUS.DELTA.X (doubleword)	Delta value for horizontal movement, in pixels. (Distance the pointer must move horizontally in order to cause a movement event.)		
?PTRDEV_GSTATUS.DELTA.Y (doubleword)	Delta value for vertical movement, in pixels. (Distance the pointer must move vertically in order to cause a movement event.)		
?PTRDEV_GSTATUS.EVTS	Flag word for pointer device events that can occur. See Table 2-174.		
?PTRDEV_GSTATUS.BTNS	Flag word to indicate the buttons that the button-related events apply to, as follows.		
	?PTRDEV_GSTATUS.BTNS.ONE	If this flag is set you will receive events for button 1.	
	?PTRDEV_GSTATUS.BTNS.TWO	If this flag is set you will receive events for button 2.	
	?PTRDEV_GSTATUS.BTNS.THREE	If this flag is set you will receive events for button 3.	
?PTRDEV_GSTATUS.PTR_SHAPE	The current pointer shape. Possible shapes are		
	?PTRDEV_PTR_SHAPE_DEFAULT	The default pointer shape.	
	?PTRDEV_PTR_SHAPE_ARROW	An arrow.	
	?PTRDEV_PTR_SHAPE_FINGER	A pointing finger.	
	?PTRDEV_PTR_SHAPE_SXHAIR	A small cross-hair.	
	?PTRDEV_PTR_SHAPE_FXHAIR	A full-screen cross-hair.	
	?PTRDEV_PTR_SHAPE_HOURGLASS	An hourglass.	

(continued)

Table 2–173. Contents of the ?PTRDEV_GET_PTR_STATUS Subpacket

Offset	Contents
?PTRDEV_GSTATUS.PTR_STATE	Whether the pointer is visible (enabled) or invisible (disabled) and whether digitizing is enabled or disabled. By default, the pointer is visible (enabled). ?PTRDEV_PTR_STATE_ENABLE The graphics pointer is enabled in this window and digitizing is disabled. The OS returns pointer events for the scaled area of a tablet. ?PTRDEV_PTR_STATE_DISABLE The graphics pointer is disabled in this window and digitizing is disabled. The OS returns pointer events for the scaled area of a tablet. ?PTRDEV_PTR_STATE_DIGITIZE Digitizing is enabled for the target window. If it is the upfront window, then the graphics pointer does not appear on the entire screen and the OS does not return pointer events for the scaled portion of the tablet. The OS treats any event generated over the ENTIRE tablet as an absolute tablet event. If the target window is not the upfront window, then it receives no pointer events and the graphics pointer is unaffected.

(concluded)

If a flag in Table 2–174 is set you will receive events of the corresponding type. See offset ?PTRDEV_GSTATUS.EVTS in Table 2–173.

Table 2–174. Flags for Each Possible Pointer Device Event (Flag Word ?PTRDEV_GSTATUS.EVTS)

Flag	Description
?PTRDEV_GSTATUS.EVTS.MOVEMENT	If this flag is set you will receive movement events that occur in this window. (A movement event occurs if the pointer moves while it is within the window.)
?PTRDEV_GSTATUS.EVTS.WINDOW	If this flag is set you will receive window events that occur in this window. There are two kinds of window events: o Window edge o Window activation
?PTRDEV_GSTATUS.EVTS.BTN_DOWN	If this flag is set you will receive button-down events for this window. (A button-down event occurs if the user presses a button while the pointer is within the window.)
?PTRDEV_GSTATUS.EVTS.BTN_UP	If this flag is set you will receive button-up events for this window. (A button-up event occurs if the user releases a button while the pointer is within the window.)
?PTRDEV_GSTATUS.EVTS.DBL_CLICK	If this flag is set you will receive double-click events for this window. (A double-click event occurs when the user presses and releases a button twice quickly.)
?PTRDEV_GSTATUS.EVTS.TAB_MOVEMENT	If this flag is set you will receive tablet movement events in absolute coordinates.
?PTRDEV_GSTATUS.EVTS.TAB_BTN_DOWN	If this flag is set you will receive tablet button-down events in absolute coordinates.
?PTRDEV_GSTATUS.EVTS.TAB_BTN_UP	If this flag is set you will receive tablet button-up events in absolute coordinates.
?PTRDEV_GSTATUS.EVTS.TAB_DBL_CLICK	If this flag is set you will receive tablet double-click events in absolute coordinates.

Moving the Pointer

You can use the ?PTRDEV_GENERATE_EVENT function to move the pointer within a window in the active group. The system moves the pointer to the location you specify, and places a pointer event in your input buffer that describes the pointer movement.

For example, suppose you use this function to move the pointer to location (50,50) in your application's window. In addition to moving the pointer, the system places a pointer event in the window's input buffer. This event is identical to a normal pointer event, except that it was generated via a system call instead of being caused by a user's action. The event tells your application that a movement event has taken place in your window, and that the pointer is now at location (50,50).

Because the pointer events you generate with this function are identical to those generated by a user's action with a physical pointer device, you can also use ?PTRDEV_GENERATE_EVENT to simulate pointer movement and button clicks. ?PTRDEV_GENERATE_EVENT is particularly useful for demonstrating and/or testing programs that read and interpret pointer event data.

NOTE: If the user has an absolute pointer device, such as a bit pad, ?PTRDEV_GENERATE_EVENT can change the pointer position only temporarily. Although the pointer does move to the position you specify, when the user next moves the pointer device, the pointer jumps back to the absolute position indicated by the pointer device.

In the main ?PTRDEVICE packet, specify the window for which you want to generate a pointer event, and provide a word pointer to the ?PTRDEV_GENERATE_EVENT subpacket.

In the ?PTRDEV_GENERATE_EVENT subpacket, you specify

- The type of pointer event you want to generate: movement, button up, button down, or double click.
- The coordinates to which you are moving the pointer (if you are generating a movement event).
- Whether the coordinates you are specifying are relative to the origin of the virtual terminal or to the origin of the physical screen.
- The button number (if you are generating a button–related event).

Figure 2–194 shows the subpacket structure; Table 2–175 details its contents.

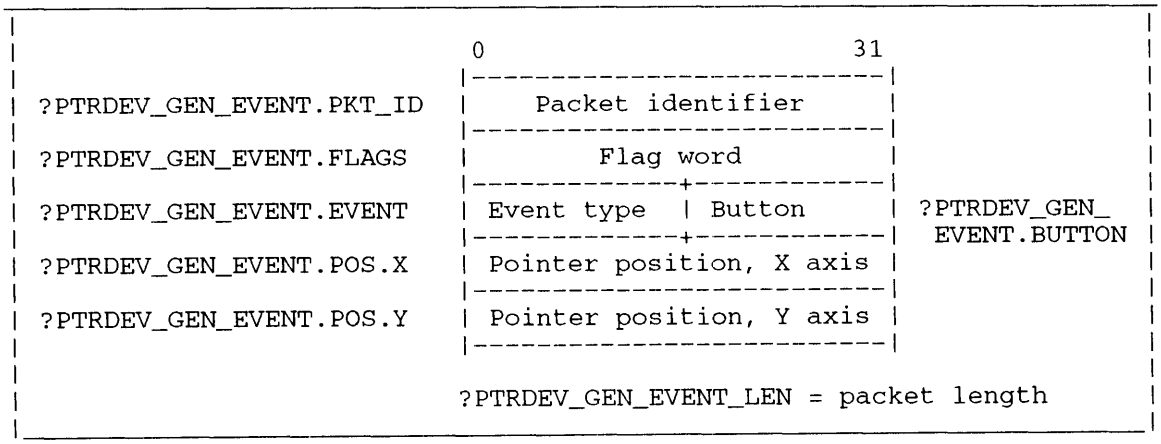


Figure 2–194. Structure of the ?PTRDEV_GENERATE_EVENT Subpacket

Table 2–175. Contents of the ?PTRDEV_GENERATE_EVENT Subpacket

Offset	Contents
?PTRDEV_GEN_EVENT.PKT_ID (doubleword)	Packet identifier: set to ?PTRDEV_GENERATE_EVENT_PKTID.
?PTRDEV_GEN_EVENT.FLAGS (doubleword)	Flag word. Flags are ?PTRDEV_GEN_EVENT.FLAGS.X -- Set this flag if you are specifying a new X coordinate for the pointer position. If you are not changing the X coordinate, set both this flag and the offset to 0. ?PTRDEV_GEN_EVENT.FLAGS.Y -- Set this flag if you are specifying a new Y coordinate for the pointer position. If you are not changing the Y coordinate, set both this flag and the offset to 0. ?PTRDEV_GEN_EVENT.VIRTUAL -- Set this flag if you are specifying a pointer position relative to the origin of the virtual terminal. (The position you specify on the virtual terminal must be visible.) ?PTRDEV_GEN_EVENT.PHYSICAL -- Set this flag if you are specifying a pointer position relative to the origin of the physical screen.
?PTRDEV_GEN_EVENT.EVENT	The type of event you want to generate. Event types are ?PTRDEV_EVENTS_MOVEMENT Movement ?PTRDEV_EVENTS_BTN_UP Button up ?PTRDEV_EVENTS_BTN_DOWN Button down ?PTRDEV_EVENTS_DBL_CLICK Double click
?PTRDEV_GEN_EVENT.BUTTON	The number of the button (if any) involved in the event. For a movement event, you must set this field to 0.
?PTRDEV_GEN_EVENT.POS.X (doubleword)	X coordinate of the desired pointer position. If relative to the origin of the physical screen, specify in pixels. If relative to the origin of the virtual terminal, specify in units appropriate to the virtual terminal type.
?PTRDEV_GEN_EVENT.POS.Y (doubleword)	Y coordinate of the desired pointer position. If relative to the origin of the physical screen, specify in pixels. If relative to the origin of the virtual terminal, specify in units appropriate to the virtual terminal type.

Getting the Location and Status of the Pointer

You use ?PTRDEVICE, function code ?PTRDEV_GET_PTR_LOCATION, to find out the current status of the pointer. This function returns

- The current location of the pointer within the specified window.
- The current status of the pointer device buttons.

In the main ?PTRDEVICE packet, specify the window for which you want to get the pointer's location and button status, and provide a word pointer to the ?PTRDEV_GET_PTR_LOCATION subpacket. (You can specify only one window at a time; if the pointer is not within that window, this function returns an error.) The operating system returns the pointer's location and button status in the subpacket.

Figure 2–195 shows the structure of the ?PTRDEV_GET_PTR_LOCATION subpacket; Table 2–176 details its contents.

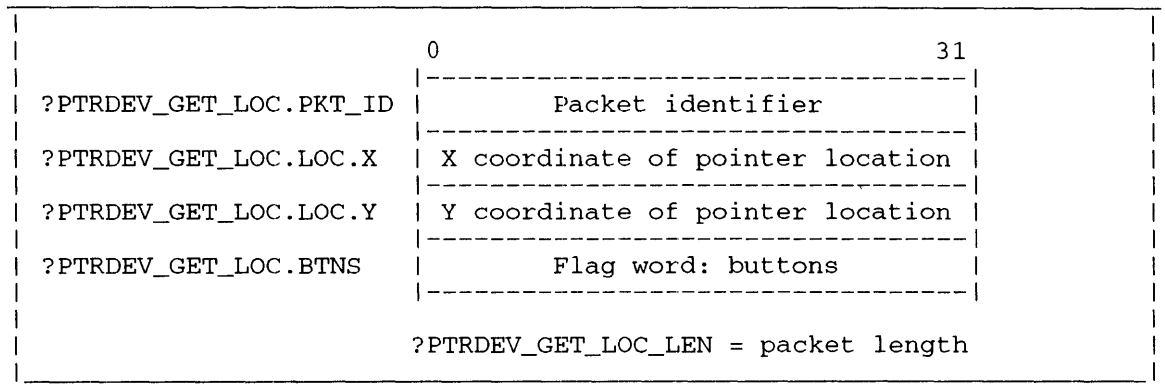


Figure 2–195. Structure of the ?PTRDEV_GET_PTR_LOCATION Subpacket

Table 2–176. Contents of the ?PTRDEV_GET_PTR_LOCATION Subpacket

Offset	Contents
?PTRDEV_GET_LOC.PKT_ID (doubleword)	Packet identifier. Place ?PTRDEV_GET_LOC_PKTID here.
?PTRDEV_GET_LOC.LOC.X (doubleword)	X coordinate of pointer location within the window specified in the main packet. The coordinate value is in characters for a character window, in pixels for a graphics window.
?PTRDEV_GET_LOC.LOC.Y (doubleword)	Y coordinate of pointer location within the window specified in the main packet. The coordinate value is in characters for a character window, in pixels for a graphics window.
?PTRDEV_GET_LOC.BTNS (doubleword)	Flag word. Indicates which buttons on the pointer device are currently down. For each button that is down, the operating system sets the bit that corresponds to the button number. (For example, if Button 2 is down, the operating system sets Bit 2 of the flag word.)

Getting the Tablet Status

You use ?PTRDEVICE with function code ?PTRDEV_GET_TABLET_LOCATION to poll the puck or pen position and state. The position that the operating system returns is the absolute location of the pointing device on the tablet. You might or might not see the cursor on the screen.

Function code ?PTRDEV_GET_TABLET_LOCATION yields results that are quite similar to those of function code ?PTRDEV_GET_PTR_LOCATION. A significant difference is the values of the locations that the operating system returns. The former code returns the absolute location of the pointer device on the tablet with a resolution for both x and y coordinates between 0 and 2793. The state of digitizing (enabled or disabled) has no effect on this code. The latter code returns the position of the cursor relative to the active window in virtual coordinates.

The operating system returns error ERIPP (invalid pointer device parameter) if you issue ?PTRDEVICE with function code ?PTRDEV_GET_TABLET_LOCATION and with a relative device such as a mouse.

In the main ?PTRDEVICE packet, specify the window for which you want to get the pointing device's absolute location on the tablet, and provide a word pointer to the ?PTRDEV_GET_TABLET_LOCATION subpacket. You can specify only one window at a time. The operating system returns the locations (absolute x and y coordinates) in the subpacket.

Figure 2–196 shows the structure of the ?PTRDEV_GET_TABLET_LOCATION subpacket; Table 2–177 details its contents.

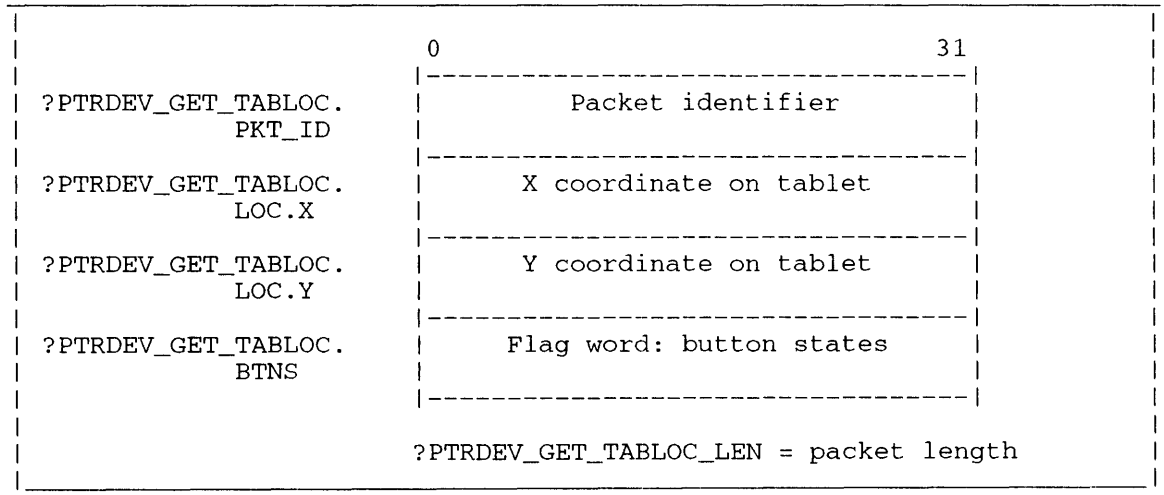


Figure 2–196. Structure of the ?PTRDEV_GET_TABLET_LOCATION Subpacket

Table 2–177. Contents of the ?PTRDEV_GET_TABLET_LOCATION Subpacket

Offset	Contents
?PTRDEV_GET_TABLOC.PKT_ID (doubleword)	Packet identifier. Place ?PTRDEV_GET_TABLOC_PKTID here.
?PTRDEV_GET_TABLOC.LOC.X (doubleword)	X coordinate of the pointer location on the tablet. The value is an absolute coordinate.
?PTRDEV_GET_TABLOC.LOC.Y (doubleword)	Y coordinate of the pointer location on the tablet. The value is an absolute coordinate.
?PTRDEV_GET_TABLOC.BTNS (doubleword)	Flag word. Indicates which buttons on the pointer device are currently down. For each button that is down, the operating system sets the bit that corresponds to the button number. (For example, if Button 2 is down, the operating system sets Bit 2 of the flag word.)

Notes

- See the description of ?GECHR in this chapter.

?PWDCRYP Performs a password data encryption request.

AOS/VS

?PWDCRYP *[packet address]*
error return
normal return

Input

AC0	Reserved (Set to 0.)
AC1	Reserved (Set to 0.)
AC2	Address of the ?PWDCRYP packet unless you specify the address as an argument to ?PWDCRYP

Output

AC0	Undefined
AC1	Unchanged
AC2	Unchanged

Error Codes in AC0

ERVBP	Invalid byte pointer passed as an argument
ERRVN	Reserved value not zero
ERPVS	Packet revision not supported
ERPRE	Invalid parameter passed as system call argument

Why Use It?

You can use ?PWDCRYP to add a level of security to the system password file. By using the ?PWDCRYP system call, the plain text password input to the call will be encrypted and returned to the caller. This system call could be used by products wishing secure system storage of various user profiles at a higher level functional layer than that of the operating system.

Who Can Use It?

There are no special process privileges needed to issue this call, and there are no restrictions concerning file access.

What It Does

?PWDCRYP performs a one way password encryption which uses the password as both the plain text and the "key" in the algorithm. The algorithm is based on the Data Encryption Standard (DES) algorithm proposed by the National Bureau of Standards (NBS).

?PWDCRYP performs the following function on behalf of the calling process:

- Perform a one–way password encryption of a plain text password whose byte address is contained in offset ?PWBP of the packet. The candidate password must be 1 to 32 bytes in length and be null terminated in the buffer. (EXEC requires passwords to contain between 6 and 15 bytes, but you can use ?PWDCRYP on longer passwords in your applications.)

The encrypted password is a multiple of 8 bytes long. If, for example, the input is 9 bytes plus a null terminator then the output is 16 bytes long; ?PWDCRYP has encrypted the null terminator.

We recommend that you pad *all* passwords to 16 bytes (using ASCII <377> as the pad character) and supply the null character in the 17th byte. For example, the recommended way to supply the password WESTBOROUGH is

WESTBOROUGH<377><377><377><377><377><000>

Before issuing ?PWDCRYP, set the flags word (?PWFW) according to the function to be performed (see Table 2–178), and set the remaining offsets appropriately (see Figure 2–197). The operating system will return the length in bytes of the null terminated encrypted password. The length of the packet is ?PWSZ words.

Figure 2–197 shows the structure of the ?PWDCRYP packet, and Table 2–178 describes the contents of each offset.

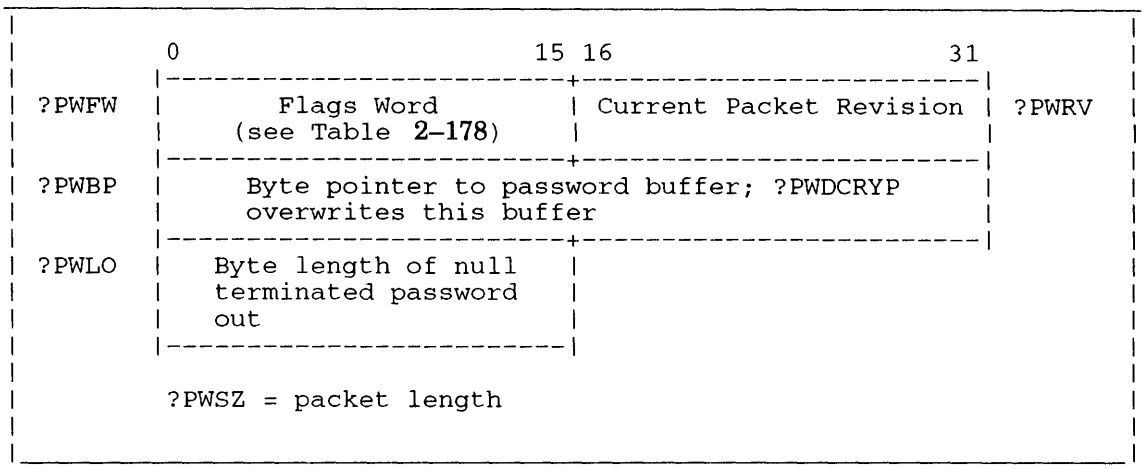


Figure 2–197. Structure of ?PWDCRYP Packet

Table 2–178. Contents of ?PWDCRYP Packet

Offset	Contents
?PWFW	Flags word contains the following flags: ?PWOW--perform a one-way password encryption (Flag ?PWOW must be set on input to ?PWDCRYP.)
?PWRV	Current packet revision number; should be zero.
?PWBP	Byte address of password buffer containing the password in your logical address space. NOTE: The password buffer length must be at least 33 bytes in length.
?PWLO	Length in bytes of the output password that is null terminated. The OS returns this value.

?PWRB — Performs physical block I/O.

?PWRB *[packet address]*
error return
normal return

See the description of ?PRDB in this chapter for further information on ?PWRB.

?R through ?Z

Descriptions of the system calls whose first letters begin with ?R through ?Z are in Chapter 2 of the manual *AOS/VS, AOS/VS II, and AOS/RT32 System Call Dictionary, ?R Through ?Z* (093–000543).

End of System Calls ?A Through ?Q

Index

Within the index, a bold page number indicates a primary reference. A range of page numbers indicates the reference spans those pages. A reference such as ?RTODC_PKT... indicates that all references beginning with those characters are found on the referenced pages.

Symbols

Numbers

A

B

C

D

E

F

086–000195 updates
093–000542

G

H

I

J

K

L

M

N

O

P

Q

R

S

086–000195 updates
093–000542

T

U

V

W

X

Y

Z

086–000195 updates
093–000542

Document Set

For Users

AOS/VS and AOS/VS II Glossary (069–000231)

For all users, this manual defines important terms used in AOS/VS and AOS/VS II manuals, both regular and preinstalled.

Learning to Use Your AOS/VS System (069–000031)

A primer for all users, this manual introduces AOS/VS (but the material applies to AOS/VS II) through interactive sessions with the CLI, the SED and SPEED text editors, programming languages, Assembler, and the Sort/Merge utility.
Using the CLI (AOS and AOS/VS) is a good follow–up.

SED Text Editor User's Manual (AOS and AOS/VS) (093–000249)

For all users, this manual explains how to use SED, an easy–to–use screen–oriented text editor that lets you program function keys to make repetitive tasks easier. The *SED Text Editor* template (093–000361) accompanies this manual.

Using the AOS/VS System Management Interface (SMI) (069–000203)
Using the AOS/VS II System Management Interface (SMI) (069–000311)

For those working with preinstalled systems and those on regular systems who want an alternative to the CLI, the SMI is an easy–to–use, menu–driven program that helps with system management functions and some file maintenance tasks.

Using the CLI (AOS/VS and AOS/VS II) (093–000646)

For all users, this manual explains the AOS/VS and AOS/VS II file and directory structure and how to use the CLI, a command line interpreter, as the interface to the operating system. This manual explains how to use the CLI macro facility, and includes a dictionary of CLI commands and pseudomacros.

For System Managers and Operators

AOS/VS and AOS/VS II Error and Status Messages (093–000540)

For all users, but especially for system managers and operators of regular systems, this manual lists error and status messages, their source and meaning, and appropriate responses. This manual complements *Installing, Starting, and Stopping AOS/VS*; *Installing, Starting, and Stopping AOS/VS II*; and *Managing AOS/VS and AOS/VS II*.

AOS/VS and AOS/VS II Menu–Based Utilities (093–000650)

A keyboard template to identify function keys. A number of system management programs—such as Disk Jockey, VSGEN, and the SMI—and the BROWSE utility use the function keys identified on this template.

Information Update: Starting Your ECLIPSE MV/1000 DC (014–001728)

Updates *Starting and Updating Preinstalled AOS/VS* and *Starting and Updating Preinstalled AOS/VS II.*

Installing, Starting, and Stopping AOS/VS (093–000675)
Installing, Starting, and Stopping AOS/VS II (093–000539)

For system managers and operators of regular (as opposed to preinstalled) systems, these manuals explain the steps necessary to format disks, install a tailored operating system, create the multiuser environment, update the system or microcode, and routinely start up and shut down the system. *AOS/VS and AOS/VS II Error and Status Messages* and *Managing AOS/VS and AOS/VS II* are companions to these manuals.

Managing AOS/VS and AOS/VS II (093–000541)

For system managers and operators, this manual explains managing an AOS/VS or AOS/VS II system. Managing tasks include such topics as editing user profiles, managing the multiuser environment with the EXEC program, backing up and restoring files, using runtime tools, and so forth. This manual complements the "Installing" manuals, whether for regular or preinstalled systems.

Starting and Updating Preinstalled AOS/VS (069–000293)
Starting and Updating Preinstalled AOS/VS II (069–000294)

For those working with preinstalled (as opposed to regular) operating systems on all computers except ECLIPSE MV/3500™ DC and MV/5000 Series systems, these manuals explain how to start, update, and change certain system parameters. The manuals also help you interpret error messages and codes. Companion manuals are *Using the AOS/VS System Management Interface* and *Using the AOS/VS II System Management Interface.*

Starting and Updating Preinstalled AOS/VS on ECLIPSE MV/3500™ DC and MV/5000™ DC Series Systems (069–000481)
Starting and Updating Preinstalled AOS/VS II on ECLIPSE MV/3500™ DC and MV/5000™ DC Series Systems (069–000480)

For those working with preinstalled (as opposed to regular) operating systems on ECLIPSE MV/3500™ DC and MV/5000™ DC Series computers, these manuals explain how to start, update, and change certain system parameters. The manuals also help you interpret error messages and codes. Companion manuals are *Using the AOS/VS System Management Interface* and *Using the AOS/VS II System Management Interface.*

If you have one of these computer systems, use the pertinent manual above; discard any other *Starting and Updating Preinstalled* manuals you receive.

Using the AOS/VS System Management Interface (SMI) (069–000203)
Using the AOS/VS II System Management Interface (SMI) (069–000311)

For those working with preinstalled systems and those on regular systems who want an alternative to the CLI, the SMI is an easy–to–use, menu–driven program that helps with system management functions and some file maintenance tasks.

For Programmers

AOS/VS, AOS/VS II, and AOS/RT32 System Call Dictionary, ?A through ?Q (093–000542)
AOS/VS, AOS/VS II, and AOS/RT32 System Call Dictionary, ?R through ?Z (093–000543)

For system programmers and application programmers who use system calls, this two–volume manual provides detailed information about system calls, including their use, syntax, accumulator input and output values, parameter packets, and error codes. *AOS/VS System Concepts* is a companion manual.

AOS/VS Debugger and File Editor User's Manual (093–000246)

For assembly language programmers, this manual describes using the AOS/VS and AOS/VS II debugger for examining program files, and the file editor FED for examining and modifying locations in any kind of disk file, including program and text files. The *AOS/VS Debug/FED* template (093–000396) accompanies this manual.

AOS/VS Link and Library File Editor (LFE) User's Manual (093–000245)

For AOS/VS and AOS/VS II programmers, this manual describes the Link utility, which builds executable program files from object modules and library files, and which can also be used to create programs to run under the AOS, MP/AOS, RDOS, RTOS, or DG/UX™ operating systems. This manual also describes the Library File Editor utility, LFE, for creating, editing, and analyzing library files; and the utilities CONVERT and MKABS, for manipulating RDOS and RTOS files.

AOS/VS Macroassembler (MASM) Reference Manual (093–000242)

For assembly language programmers, this reference manual describes the use and operation of the MASM utility, which works under AOS/VS and AOS/VS II.

AOS/VS System Concepts (093–000335)

For system programmers and application programmers who write assembly–language subroutines, this manual explains basic AOS/VS system concepts, most of which apply to AOS/VS II as well. This manual complements both volumes of the *AOS/VS, AOS/VS II, and AOS/RT32 System Call Dictionary.*

SPEED Text Editor (AOS and AOS/VS) User's Manual (093–000197)

For programmers, this manual explains how to use SPEED, a powerful (but unforgiving) character–oriented text editor.

Other Related Documents

AOS/VS and AOS/VS II Performance Package User's Manual (093–000364)

For system managers, this manual explains how to use the AOS/VS and AOS/VS II Performance Package (Model 30718), a separate product that is useful for analyzing and perhaps improving the performance of AOS/VS and AOS/VS II systems.

Backing Up and Restoring Files With DUMP_3/LOAD_3 (093–000561)

For system managers, operators, and experienced users, this manual explains the DUMP_3/LOAD_3 product, separately available, which provides backup and enhanced restoration functions, including precise indexing of files on a backup tape set.

Configuring and Managing the High-Availability Disk-Array/MV (H.A.D.A/MV) Subsystem (014–002160)

For system managers of the H.A.D.A./MV subsystem (a separate product), this manual explains how to configure, operate, and replace subsystem controllers, disk modules, and tape modules. This manual also explains how to replace fans, power supplies, and other subsystem hardware.

Configuring Your Network with XTS (093–00689)

For network administrators, managers, or operators responsible for designing, configuring, or maintaining a network management system, this manual describes how to manage and operate Data General's XODIAC™ Transport Service (XTS and XTS II) under AOS/VS and AOS/VS II.

Installing and Administering DG TCP/IP (093–701027)

For network managers and operators, this manual explains how to install and manage a TCP/IP network under AOS/VS.

Managing AOS/VS II ONC™ /NFS® Services (093–000667)

For network managers and operators, this manual explains how to install and manage an ONC Network File server software under AOS/VS II.

Managing AOS/VS II TCP/IP (093–000704)

For network managers and operators, this manual explains how to install and manage a TCP/IP network under AOS/VS II.

Managing and Operating the XODIAC™ Network Management System (093–000260)

For network managers and operators, this manual describes how to install and manage the Data General proprietary network software.

Managing XTS II with DG/OpenNMS (093–000698)

For network managers and operators, this manual explains how to use DG/OpenNMS to manage the XTS II transport service for large communications networks. It also identifies the XTS II components and explains how to use the NMI menus and screens to manage the XTS II subsystems and the Message Transport Agent (MTA).

*Managing Your DG/PC*Integration Network with DG/ONMS* (093–000624)

For network managers, this manual explains how to manage XTS II and DG/PC*Integration components with DG/OpenNMS.

Managing Your Network with DG/OpenNMS (093–000486)

For network managers, administrators, and operators, this manual describes how to use the DG/OpenNMS software. It also explains how to load the software, create the DG/OpenNMS environment, and use the Network Management Interface (NMI) to manage the network.

Managing Your XODIAC™ Network with DG/ONMS (093–000625)

For network managers, this manual explains how to manage XTS II, MTA, and the XODIAC agents (FTA, RMA, and SVTA) with DG/OpenNMS.

Programming with the Remote Procedure Call (RPC) on AOS/VS II (093–000770)

For experienced network programmers, this manual provides information necessary to write the Remote Procedure Call for the AOS/VS II UDP/IP and TCP/IP networks.

Using CLASP (Class Assignment and Scheduling Package) (093–000422)

For system managers, this manual explains how to use the AOS/VS and AOS/VS II Class Assignment and Scheduling Package (Model 31134), a separate product that is useful for tailoring process scheduling to the needs of a specific site.

Using the MV Data Center Manager (093–000769)

For system managers, this manual explains how to use the MV Data Center Manager software, a separate product that manages multiple ECLIPSE MV/Family computers from an AViiON workstation.

End of Document Set

TO ORDER

1. An order can be placed with the TIPS group in two ways:
   a) MAIL ORDER – Use the order form on the opposite page and fill in all requested information. Be sure to include shipping charges and local sales tax. If applicable, write in your tax exempt number in the space provided on the order form.

      Send your order form with payment to:
      Data General Corporation
      ATTN: Educational Services/TIPS G155
      4400 Computer Drive
      Westboro, MA 01581–9973

   b) TELEPHONE – Call TIPS at (508) 870–1600 for all orders that will be charged by credit card or paid for by purchase orders over $50.00. Operators are available from 8:30 AM to 5:00 PM EST.

METHOD OF PAYMENT

2. As a customer, you have several payment options:
   a) Purchase Order – Minimum of $50. If ordering by mail, a hard copy of the purchase order must accompany order.
   b) Check or Money Order – Make payable to Data General Corporation.
   c) Credit Card – A minimum order of $20 is required for MasterCard or Visa orders.

SHIPPING

3. To determine the charge for UPS shipping and handling, check the total quantity of units in your order and refer to the following chart:

Total Quantity	Shipping & Handling Charge
1–4 Items	$5.00
5–10 Items	$8.00
11–40 Items	$10.00
41–200 Items	$30.00
Over 200 Items	$100.00

   If overnight or second day shipment is desired, this information should be indicated on the order form. A separate charge will be determined at time of shipment and added to your bill.

VOLUME DISCOUNTS

4. The TIPS discount schedule is based upon the total value of the order.

Order Amount	Discount
$0–$149.99	0%
$150–$499.99	10%
Over $500	20%

TERMS AND CONDITIONS

5. Read the TIPS terms and conditions on the reverse side of the order form carefully. These must be adhered to at all times.

DELIVERY

6. Allow at least two weeks for delivery.

RETURNS

7. Items ordered through the TIPS catalog may not be returned for credit.
8. Order discrepancies must be reported within 15 days of shipment date. Contact your TIPS Administrator at (508) 870–1600 to notify the TIPS department of any problems.

INTERNATIONAL ORDERS

9. Customers outside of the United States must obtain documentation from their local Data General Subsidiary or Representative. Any TIPS orders received by Data General U.S. Headquarters will be forwarded to the appropriate DG Subsidiary or Representative for processing.

TIPS ORDER FORM

Mail To: Data General Corporation
Attn: Educational Services/TIPS G155
4400 Computer Drive
Westboro, MA 01581 - 9973

BILL TO:	SHIP TO: (No P.O. Boxes - Complete Only if Different Address)
COMPANY NAME _____	COMPANY NAME _____
ATTN: _____	ATTN: _____
ADDRESS _____	ADDRESS (NO PO BOXES) _____
CITY _____	CITY _____
STATE _____ ZIP _____	STATE _____ ZIP _____

Priority Code _____ (See label on back of catalog)

Authorized Signature of Buyer (Agrees to terms & conditions on reverse side)	Title	Date	Phone (Area Code)	Ext.

ORDER #	QTY	DESCRIPTION	UNIT PRICE	TOTAL PRICE

A SHIPPING & HANDLING

☐ UPS	ADD
1–4 Items	$5.00
5–10 Items	$8.00
11–40 Items	$10.00
41–200 Items	$30.00
200+ Items	$100.00

Check for faster delivery

Additional charge to be determined at time of shipment and added to your bill.

☐ UPS Blue Label (2 day shipping)
☐ Red Label (overnight shipping)

B VOLUME DISCOUNTS

Order Amount	Save
$0–$149.99	0%
$150–$499.99	10%
Over $500.00	20%

Tax Exempt # or Sales Tax (if applicable) _____

ORDER TOTAL	
Less Discount See B	–
SUB TOTAL	
Your local* sales tax	+
Shipping and handling – See A	+
TOTAL – See C	

C PAYMENT METHOD

☐ Purchase Order Attached ($50 minimum)
P.O. number is _____. (Include hardcopy P.O.)
☐ Check or Money Order Enclosed
☐ Visa ☐ MasterCard ($20 minimum on credit cards)

Account Number ☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐ Expiration Date ☐☐☐☐

_____
Authorized Signature
(Credit card orders without signature and expiration date cannot be processed.)

THANK YOU FOR YOUR ORDER

PRICES SUBJECT TO CHANGE WITHOUT PRIOR NOTICE.
PLEASE ALLOW 2 WEEKS FOR DELIVERY.
NO REFUNDS NO RETURNS.

* Data General is required by law to collect applicable sales or use tax on all purchases shipped to states where DG maintains a place of business, which covers all 50 states. Please include your local taxes when determining the total value of your order. If you are uncertain about the correct tax amount, please call 508–870–1600.

Form 702
Rev. 8/87

DATA GENERAL CORPORATION
TECHNICAL INFORMATION AND PUBLICATIONS SERVICE
TERMS AND CONDITIONS

Data General Corporation ("DGC") provides its Technical Information and Publications Service (TIPS) solely in accordance with the following terms and conditions and more specifically to the Customer signing the Educational Services TIPS Order Form. These terms and conditions apply to all orders, telephone, telex, or mail. By accepting these products the Customer accepts and agrees to be bound by these terms and conditions.

1. CUSTOMER CERTIFICATION
Customer hereby certifies that it is the owner or lessee of the DGC equipment and/or licensee/sub–licensee of the software which is the subject matter of the publication(s) ordered hereunder.

2. TAXES
Customer shall be responsible for all taxes, including taxes paid or payable by DGC for products or services supplied under this Agreement, exclusive of taxes based on DGC's net income, unless Customer provides written proof of exemption.

3. DATA AND PROPRIETARY RIGHTS
Portions of the publications and materials supplied under this Agreement are proprietary and will be so marked. Customer shall abide by such markings. DGC retains for itself exclusively all proprietary rights (including manufacturing rights) in and to all designs, engineering details and other data pertaining to the products described in such publication. Licensed software materials are provided pursuant to the terms and conditions of the Program License Agreement (PLA) between the Customer and DGC and such PLA is made a part of and incorporated into this Agreement by reference. A copyright notice on any data by itself does not constitute or evidence a publication or public disclosure.

4. LIMITED MEDIA WARRANTY
DGC warrants the CLI Macros media, provided by DGC to the Customer under this Agreement, against physical defects for a period of ninety (90) days from the date of shipment by DGC. DGC will replace defective media at no charge to you, provided it is returned postage prepaid to DGC within the ninety (90) day warranty period. This shall be your exclusive remedy and DGC's sole obligation and liability for defective media. This limited media warranty does not apply if the media has been damaged by accident, abuse or misuse.

5. DISCLAIMER OF WARRANTY
EXCEPT FOR THE LIMITED MEDIA WARRANTY NOTED ABOVE, DGC MAKES NO WARRANTIES, EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, WARRANTIES OF MERCHANTABILITY AND FITNESS FOR PARTICULAR PURPOSE ON ANY OF THE PUBLICATIONS, CLI MACROS OR MATERIALS SUPPLIED HEREUNDER.

6. LIMITATION OF LIABILITY
A. CUSTOMER AGREES THAT DGC'S LIABILITY, IF ANY, FOR DAMAGES, INCLUDING BUT NOT LIMITED TO LIABILITY ARISING OUT OF CONTRACT, NEGLIGENCE, STRICT LIABILITY IN TORT OR WARRANTY SHALL NOT EXCEED THE CHARGES PAID BY CUSTOMER FOR THE PARTICULAR PUBLICATION OR CLI MACRO INVOLVED. THIS LIMITATION OF LIABILITY SHALL NOT APPLY TO CLAIMS FOR PERSONAL INJURY CAUSED SOLELY BY DGC'S NEGLIGENCE. OTHER THAN THE CHARGES REFERENCED HEREIN, IN NO EVENT SHALL DGC BE LIABLE FOR ANY INCIDENTAL, INDIRECT, SPECIAL OR CONSEQUENTIAL DAMAGES WHATSOEVER, INCLUDING BUT NOT LIMITED TO LOST PROFITS AND DAMAGES RESULTING FROM LOSS OF USE, OR LOST DATA, OR DELIVERY DELAYS, EVEN IF DGC HAS BEEN ADVISED, KNEW OR SHOULD HAVE KNOWN OF THE POSSIBILITY THEREOF; OR FOR ANY CLAIM BY ANY THIRD PARTY.

B. ANY ACTION AGAINST DGC MUST BE COMMENCED WITHIN ONE (1) YEAR AFTER THE CAUSE OF ACTION ACCRUES.

7. GENERAL
A valid contract binding upon DGC will come into being only at the time of DGC's acceptance of the referenced Educational Services Order Form. Such contract is governed by the laws of the Commonwealth of Massachusetts, excluding its conflict of law rules. Such contract is not assignable. These terms and conditions constitute the entire agreement between the parties with respect to the subject matter hereof and supersedes all prior oral or written communications, agreements and understandings. These terms and conditions shall prevail notwithstanding any different, conflicting or additional terms and conditions which may appear on any order submitted by Customer. DGC hereby rejects all such different, conflicting, or additional terms.

8. IMPORTANT NOTICE REGARDING AOS/VS INTERNALS SERIES (ORDER #1865 & #1875)
Customer understands that information and material presented in the AOS/VS Internals Series documents may be specific to a particular revision of the product. Consequently user programs or systems based on this information and material may be revision–locked and may not function properly with prior or future revisions of the product. Therefore, Data General makes no representations as to the utility of this information and material beyond the current revision level which is the subject of the manual. Any use thereof by you or your company is at your own risk. Data General disclaims any liability arising from any such use and I and my company (Customer) hold Data General completely harmless therefrom.

AOS/VS, AOS/VS II, and AOS/RT32 System Call Dictionary ?A Through ?Q

093–000542–02

Cut here and insert in binder spine pocket