Data General
Data General Corporation, Westboro, Massachusetts 01580

Customer Documentation

AOS/VS, AOS/VS II, and AOS/RT32 System Call Dictionary, ?R Through ?Z

093-000543-02

AOS/VS, AOS/VS II, and AOS/RT32 System Call Dictionary, ?R Through ?Z

093–000543–02

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

Printed in the United States of America
Rev. 02, December, 1991
Licensed Material – Property of Data General Corporation
Ordering No. 093–000543

Notice

AOS/VS, AOS/VS II, and AOS/RT32 System Call Dictionary, ?R Through ?Z
093-000543-02

Revision History:
Original Release – October 1988
First Revision – February 1990
Second Revision – December 1991
Addendum 086-000196 – 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.

Data General

Addendum

Addendum to AOS/VS, AOS/VS II, and AOS/RT32 System Call Dictionary, ?R Through ?Z

086–000196–00

This addendum updates your manual 093–000543–02. 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. 086–000196

Notice

Addendum to
AOS/VS, AOS/VS II, and AOS/RT32 System Call Dictionary, ?R Through ?Z
086-000196-00

In the margins of replacement pages, a vertical bar indicates substantive technical change from 093-000543-02.

The addendum number appears on all pages in this addendum.

Updating Your Manual

This addendum (086–000196–00) to *AOS/VS, AOS/VS II, and AOS/RT32 System Call Dictionary, ?R Through ?Z* introduces new information effective with AOS/VS II Release 2.20. It also includes minor corrections.

To update your copy of 093–000543–02, please remove manual pages and insert addendum pages as follows:

Remove	Insert
Title/Notice	Title/Notice
iii/iv	iii/iv
old Table of Contents	new Table of Contents
2–595 through 2–604	2–595 through 2–604
2–625/2–626	2–625/2–626
2–629/2–630	2–629 through 2–632.2
2–643/2–644	2–643/2–644
2–649 through 2–654	2–649 through 2–654
2–693/2–694	2–693/2–694
2–703/2–704	2–703/2–704
2–709 through 2–712	2–709 through 2–712
2–721 through 2–724	2–721 through 2–724
2–741 through 2–744	2–741 through 2–744
2–759 through 2–768	2–759 through 2–768
2–783/2–784	2–783/2–784
2–861/ 2–862	2–861/ 2–862
2–877/2–878	2–877/2–878
2–881/2–882	2–881/2–882
B–5/B–6	B–5 through B–6.2
B–13/B14	B–13 through B–14.2
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 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  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

086–000196–00 updates
093–000543

?A Through ?Q

Index

Document Set

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

?R Through ?Z

Appendix A — Sample Programs

Appendix B — System Log Record Format

Index

Document Set

Tables

Table **?A Through ?Q**

?R Through ?Z

Figures

Figure **?A Through ?Q**

?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 (Continued)

This chapter describes system calls whose first letters begin with ?R through ?Z. It is a continuation of Chapter 2 in the manual *AOS/VS, AOS/VS II, and AOS/RT32 System Call Dictionary, ?A Through ?Q*.

?A through ?Q

Descriptions of system calls whose first letters begin with ?A through ?Q are in Chapter 2 in the manual *AOS/VS, AOS/VS II, and AOS/RT32 System Call Dictionary, ?A Through ?Q*. Near the beginning of Chapter 2 in the manual *AOS/VS, AOS/VS II, and AOS/RT32 System Call Dictionary, ?A Through ?Q* there is a Table 2–1. It summarizes all AOS/VS and AOS/RT32 system calls, and groups them according to their functions. It also indicates operating system differences.

?RCALL — Releases one resource and acquires a new one (16–bit processes only).

?RCALL *[procedure entry]*
normal return
alternate return

Input

Input accumulators and carry bit are passed to the new procedure

Output

Accumulators and carry bit are unchanged, unless the new procedure modifies them

Error Codes in AC0

The following error codes may be passed to the ?BOMB routine:

ERICM	Illegal system command
ERLRF	Overlay load error
ERSEN	Invalid shared library reference (The operating system does not support shared libraries.)

Why Use It?

?RCALL and the other resource system calls (?KCALL and ?RCHAIN) allow you to augment a 16–bit process's address space by acquiring disk–resident resources. You can use the resource system calls to acquire both root and overlay resources.

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

?RCALL releases the calling resource, loads a new resource, and transfers control to the procedure entry that you specify within the new resource. You can pass the procedure entry as an argument to ?RCALL or pass it on the stack (with the .PTARG pseudo–op). If you pass the procedure entry on the stack, the operating system pops it off the stack as it executes ?RCALL.

After the operating system executes the new target procedure, it returns control to the ?RCALL normal return, unless the target specifies an alternate return by issuing an

```
ISZ ?ORTN,3
```

instruction followed by a

```
RTN
```

instruction.

If the calling resource is an overlay in a multiple overlay area, it must be position–independent and movable. Otherwise, the return address may be invalid.

Notes

- See the descriptions of ?KCALL and ?RCHAIN in this chapter.

?RCHAIN Chains to a new procedure (16–bit processes only).

?RCHAIN *[procedure entry]*

Input

Input accumulators and carry bit are passed to the new procedure

Output

Accumulators and carry bit are unchanged, unless the new procedure modifies them

Error Codes

The following codes may be passed to the ?BOMB routine:

ERICM	Illegal system command
ERLRF	Overlay load error
ERSEN	Invalid shared library reference (AOS/VS and AOS/RT32 do not support shared libraries.)

Why Use It?

?RCHAIN allows you to chain from a ?KCALLed or ?RCALLed procedure to another procedure. Only procedures that have been ?KCALLed or ?RCALLed can issue ?RCHAIN. Typically, you use ?RCHAIN to connect separate sequential pieces of a large 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

Like ?RCALL, ?RCHAIN releases the calling procedure, loads a new resource, and transfers control to the procedure entry that you specify within the new resource. You can pass the procedure entry as an argument to ?RCHAIN or pass it on the top of the stack, via the .PTARG pseudo–op.

Unlike ?RCALL, ?RCHAIN is issued from one called procedure to another and, therefore, has no normal return. Although each procedure must begin with an ?RSAVE instruction, only the last procedure on the chain should end with a RTN. When the operating system executes the RTN, it transfers control back to the initial procedure, not to the ?RCHAIN caller.

Notes

- See the descriptions of ?KCALL and ?RCHAIN in this chapter.

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

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

Input

AC0	Reserved (Set to 0.)
AC1	Target file's channel number
AC2	Address of the ?RDB/?WRB 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, unless offset ?PRNH is −1 (See "Special Magnetic Tape Considerations" in this description.)
AC2	Address of the ?RDB/?WRB packet

Error Codes in AC0

ERDIO	Attempt to issue MCA direct I/O with outstanding requests
EREOF	End of file
ERILN	Illegal link number
ERIRV	Illegal retry value
ERPRO	Attempt to issue MCA request with direct I/O in progress
ERSPC	File space exhausted
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 (?WRB only)

Why Use It?

?RDB and ?WRB allow you to perform block I/O on a file. (A system call named ?BLKIO also lets you perform block I/O on a file, plus it includes functionality for reading the next allocated element in a file.)

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 ?RDB or Write access to the file before issuing ?WRB.

What It Does

?RDB and ?WRB respectively read and write blocks of data on magnetic tape, disk, or MCA links. Note that you cannot use ?RDB to read blocks into the write–protected area of your logical address space. However, you can use ?WRB to write from the write–protected area.

Before you issue ?RDB or ?WRB, load AC1 with the channel number that the operating system assigned to the target file when you opened it (?GOPEN), and define the ?RDB or ?WRB packet in your logical address space. You can load the packet address into AC2 before you issue the block I/O system call, or you can specify the packet address as an argument to ?RDB or ?WRB. The value for the length of the packet is ?PBLT. Figure 2–198 shows the structure of the block I/O packet, and Table 2–179 describes the contents of each offset.

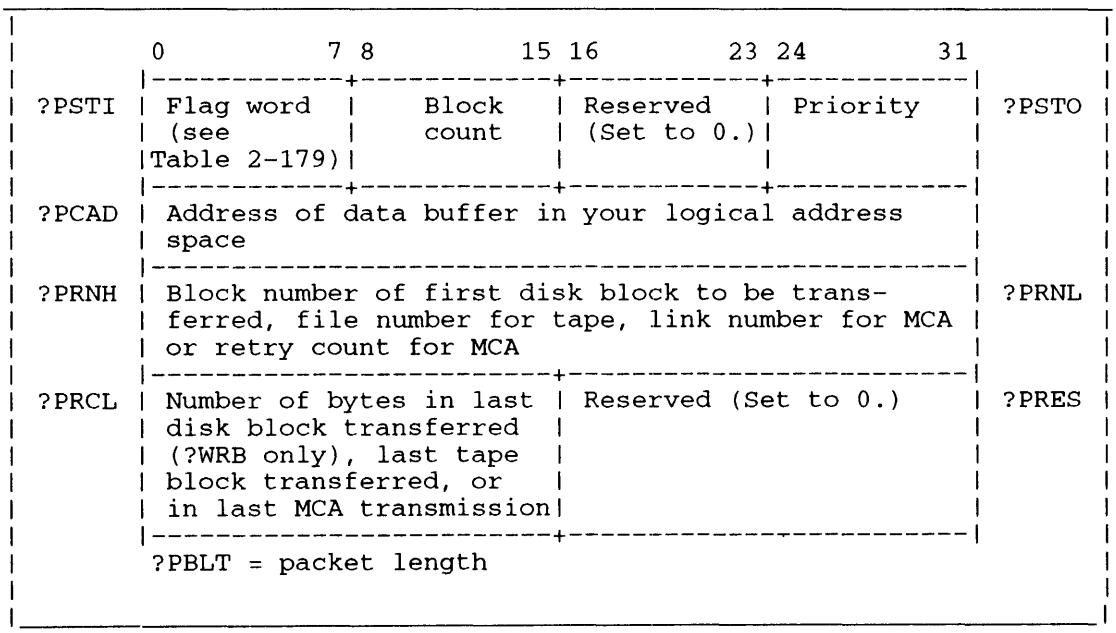


Figure 2–198. Structure of ?RDB / ?WRB Packet

Before you can issue ?RDB or ?WRB against a file, you must open it with ?GOPEN.

Table 2–179. Contents of ?RDB/?WRB Packet*

Offset	Contents
?PSTI	Left byte contains the following flags: ?IMIO--Suspend MCA protocol. ?ENOV--Override logical end-of-tape or enable vertical format unit (VFU). ?SAFM--Safe ?WRB request for magnetic tape. Right byte contains the block count.
?PSTO	Left byte is reserved. (Set to 0.) Right byte contains the priority.
?PCAD (doubleword)	Address of data buffer in your logical address space.
?PRNH	Number of first disk block transferred (high-order bits), file number for tape, or link number for MCA.
?PRNL	Low-order bits of disk file block number (?PRNH), block number or -1 for tape, or retry count for MCA.
?PRCL	Number of bytes in the last disk block transferred (for ?WRB only), number of bytes in the last tape block transferred, or the number of bytes in the last MCA transmission. Also, see the explanation of this offset in the section "Special Magnetic Tape Considerations."
?PRES	Reserved. (Set to 0.)

* There is no default unless otherwise specified.

Disk Blocks

Specify the number of blocks you want to read or write in the right byte of offset ?PSTI. Bit ?IMIO in the left byte of ?PSTI applies only to MCA transfers. Bit ?ENOV in the same offset applies only to magnetic tape transfers. For more information about these options, see "Special Magnetic Tape Considerations" and "Special MCA Considerations."

Offset ?PCAD in the packet should point 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 ?PRNH and ?PRNL to indicate the relative block number of the first disk block you want to transfer.

If you write disk blocks (?WRB), use offset ?PRCL to indicate the number of bytes in the last block you want to transfer. As in the physical block I/O packet, this value indicates the last valid byte in the block. If you issue ?WRB to extend the file, the operating system places the end–of–file mark immediately after this byte. For disk block transfers, offset ?PRCL applies only to ?WRB. 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).

Special Magnetic Tape Considerations

If you perform block I/O on magnetic tapes, you specify the file number at ?GOPEN time (for example, @MTB0:3) or in offset ?PRNH of the ?RDB/?WRB packet. If you used the CLI MOUNT

command to mount the tape, you must ?GOPEN the file using its linkname. When you specify the file number at ?GOPEN time, offset ?PRNH is ignored and offset ?PRNL contains the block number. If you do not specify a file number at ?GOPEN time, then offset ?PRNH contains the file number and ?PRNL the block number. For a 32–bit record block number use the ?B32N offset in ?BLKIO. Figure 2–1 shows the ?B32N offset on page 2–20.

When you ?GCLOSE the file, the operating system follows the last block written on a tape file with an end–of–file mark (two consecutive tape marks). By selecting the appropriate options in the block I/O packet, you can write beyond the end–of–tape mark and perform primitive tape functions, such as file positioning and rewinding.

To write beyond the end–of–tape mark, set offset ?PRNH (file number) to the highest file number plus one. Set the Block Count parameter (offset ?PSTI, right byte) to 0, and select bit ?ENOV (Bit 1) in offset ?PSTI. (Note that you must set ?ENOV to write beyond the end–of–tape mark.) Then, the operating system automatically follows the appended data with a new end–of–tape mark. To rewind a tape, set offsets ?PRNH, ?PRNL, and ?PSTI to 0.

When the operating system encounters an end–of–tape (EOT) mark while it is performing an ?RDB operation, it transfers the data it was able to read to the buffer designated by ?PCAD, and returns the byte count of that data to AC1. The byte count in AC1 is always a true value; that is, it is not rounded. The operating system then takes the error return from ?RDB, and returns error code EREOF to AC0.

Offset ?PRCL of the ?RDB/?WRB packet is the Block Length parameter for block I/O operations on magnetic tapes. If the operating system finds a block larger than this value during an ?RDB, it truncates the block to the length that you specify in ?PRCL.

The operating system always rounds the value of ?PRCL to an even number of bytes to force transfers to the buffer to be word aligned. The operating system discards the byte that exceeds the requested block size. If a block consists of an odd number of bytes on input, the operating system returns an even number of bytes on output. In this case, the last byte is undefined.

To position a tape at the end of a particular file, specify the file number in offset ?PRNH of the block I/O packet, and set offset ?PRNL (block number) to –1. To append data to the file, set the block count parameter (?PSTI, right byte) to a nonzero value. (You can append only on a ?WRB.) If you set the block count parameter to 0, the operating system returns the total number of blocks transferred to AC1.

Select the ?SAFM option in offset ?PSTI to write a logical end–of–tape (LEOT) or to write data followed by LEOT. By selecting this option, you ensure that the tape is always readable, even if you do not issue ?GCLOSE. This means that if your system fails, your tape will still be readable.

Special MCA Considerations

MCA processes in your system must be resident. You begin block I/O operations on MCAs by issuing ?GOPEN to open the MCA as a unit or to open a specific link. To open the MCA as a unit, load AC0 with a byte pointer to one of the following filenames before you issue ?GOPEN:

```
@MCAT
@MCAR
@MCAT1
@MCAR1
```

where

@MCAT	is an MCA to use for writing (transmitting).
@MCAR	is an MCA to use for reading (receiving).
1	is the second MCAT or MCAR device.

If you open the MCA as a unit, you must name a specific MCA link in offset ?PRNH of the block I/O packet for the ?RDB or ?WRB operation.

To open a specific MCA link, load AC0 with one of the following pathnames before you issue ?GOPEN:

@MCAT:n
@MCAR:n
@MCAT1:n
@MCAR1:n

where

n	is the link number (range 0 through 15).

If you open a specific MCA link, then all subsequent ?RDB or ?WRB system calls will be to that link. However, if you open the MCA as a unit, you can change the link number for each ?RDB or ?WRB system call by altering the value of ?PRNH in the ?RDB/?WRB packet. A link number of 0 applies to ?RDB system calls only, and directs the operating system to accept transmissions from any MCAT. The operating system satisfies read requests from link 0 after it performs all other read requests from specific link numbers.

When you close an MCA file previously opened for a specific link (that is, you specified the link number with ?GOPEN), the operating system transmits an end–of–file (EREOF) condition with a word count of 0 to the MCA receiver (after a 2–millisecond time–out). Do not use the link format unless the remote MCA is prepared for this transmission protocol.

The retry count, offset ?PRNL in the block I/O packet, applies only to ?WRB operations from an MCAT. You can set this parameter to any value within the range 0 through 255. Each count represents 20 attempts to establish communications with the target MCAR. A retry value of 0 directs the operating system to perform the maximum number of retries.

You can circumvent the MCA protocol and perform direct MCA I/O by selecting bit ?IMIO in offset ?PSTI. This method results in fewer interrupts. To use the direct mode, however, you must perform your own data and line validations. In addition, the operating system does not honor direct MCA transmission requests unless the MCAT's output queue is empty. If you try to transmit when the queue is not empty, the block I/O system call fails and the operating system returns error code ERPRO to AC0.

Similarly, once the operating system has queued a direct write request to an MCAT link, it does not queue any further direct I/O or protocol I/O requests until it completes the current request. If you try to perform a read or write on an MCA link while direct I/O is in progress, the system call fails, and the operating system returns error code ERDIO in AC0.

Data Channel Line Printers

If you perform block writes to a data channel line–printer controller with a DVFU unit, set bit ?ENOV in offset ?PSTI of the ?WRB packet. The DVFU (direct–access vertical forms control unit) is a hardware device that interfaces with the operating system's Forms Control Utility (FCU) program. By issuing VFU commands through the FCU, you can control the printer's horizontal and vertical formatting. A memory capability within the DVFU eliminates the need for mechanical formatting with a VFU tape.

If you omit ?ENOV, the operating system ignores all VFU definition commands.

For a complete description of the FCU utility, refer to your operating system's Command Line Interpreter (CLI) user's manual.

Sample Packet

```
PKT:    .BLK    ?PBLT           ;Allocate enough space for the packet.
                                ;Packet length = ?PBLT.
        .LOC    PKT+?PSTI       ;Number of blocks to read or write.
        .WORD   4               ;Transfer four blocks.
        .LOC    PKT+?PSTO       ;Reserved.
        .WORD   0               ;You must set this value to 0.
        .LOC    PKT+?PCAD       ;Word address of data buffer.
        .DWORD  BUFF            ;Data buffer address is BUFF.
        .LOC    PKT+?PRNH       ;Block number.
        .DWORD  0
        .LOC    PKT+?PRCL       ;Actual number of bytes transferred.
        .WORD   0               ;The OS returns this value.
        .LOC    PKT+?PRES       ;Reserved.
        .WORD   0               ;You must set this value to 0.
        .LOC    PKT+?PBLT       ;End of packet.
BUFF:   .BLK    4*256           ;Text buffer area.
```

Notes

- See the description of ?GOPEN and ?BLKIO in this chapter.

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

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

Input

AC0 One of the following:

- Byte pointer to the pathname of the file associated with the UDA
- 0 if a packet address is supplied

AC1 Reserved (Set to 0.)

AC2 One of the following:

- Address of a 128–word UDA receive buffer if not using a packet
- Address of the ?RDUDA/?WRUDA packet, unless you specify the address as an argument to ?RDUDA/?WRUDA

Output

AC0 Unchanged

AC1 Undefined

AC2 Unchanged or address of ?RDUDA/?WRUDA packet

Error Codes in AC0

ERFAD File access denied
ERIFT Illegal file type
ERNUD User data area does not exist
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_TLA_MODIFY_VIOLATION
Attempt to modify an AOS/VS II file with ?ODTL value supplied in ?GOPEN packet (?WRUDA only)

Why Use It?

?WRUDA allows you to write to a file's UDA. The UDA frequently stores forms control parameters for files you intend to print on a data channel line printer that is controlled by the EXEC utility.

(You can use a UDA for other purposes as well.) Because ?RDUDA returns the contents of a file's UDA, you can issue it to check the UDA contents before you issue ?WRUDA.

Who Can Use It?

There are no special process privileges needed to issue this call. To gain access to the file when you supply a pathname, you need Execute access to the file's parent directory and one or more of the following three access rights:

- Owner access to the file.
- Read access to the file (?RDUDA).
- Write access to the file (?WRUDA).

To gain access to the file when you supply a channel number (i.e., a packet address), you don't need Execute access to the file's parent directory — but you still need one or more of the previous three access rights.

What It Does

?WRUDA writes to the target file's user data area (UDA). ?RDUDA reads the contents of the target file's UDA. The target file or directory can be specified in one of two ways: either by a byte pointer to the entry's pathname in AC0, or by using offset ?PUDCN in the ?RDUDA/?WRUDA packet. The use of the packet is only necessary if you choose to specify the target entry's channel number. AC0 must be set to 0 to indicate the use of a packet.

If not using a packet, load AC2 with a word pointer to a 128–word receive/send buffer from/for the UDA. If using a packet, offsets ?PUDAH and ?PUDAL must instead be used. The UDA must already exist. (You can use the ?CRUDA system call or the CLI Forms Control Utility to create a UDA.)

To find out whether a file has a UDA, issue ?FSTAT, and then examine bit ?FUDA in offset ?SSTS of the returned packet. If ?FUDA is set, the UDA exists.

When you write to a UDA, do not use codes that will set Bit 0 of the first UDA word (word 0). Data General reserves codes with this configuration for its own use.

Figure 2–199 shows the structure of the ?RDUDA/?WRUDA packet.

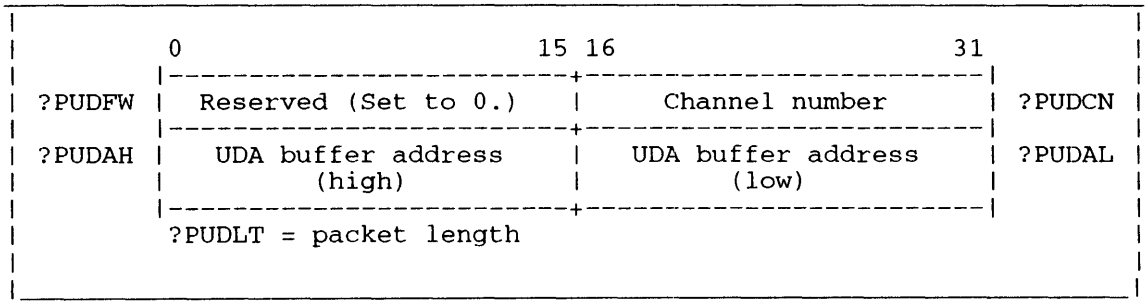


Figure 2–199. Structure of ?RDUDA/?WRUDA Packet

Notes

- See the descriptions of ?CRUDA and ?FSTAT in this chapter.

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

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

Input

AC0	Reserved (Set to 0.)
AC1	Reserved (Set to 0.)
AC2	Address of the I/O packet, unless you specify the address as an argument to the system call

Output

AC0	Undefined
AC1	Undefined
AC2	Address of the I/O packet

Error Codes in AC0

EREOF	End of file
ERFIL	File read error
ERIFL	IAC (Intelligent Asynchronous Controller) failure
ERISO	Illegal screen edit request (from PMGR)
ERITP	Translation error (Only for the selected field translation packet extension
ERLTL	Line too long
ERSPC	File space exhausted
ERPUF	Physical unit failure. If the device is a terminal, an IAC (Intelligent Asynchronous Controller) failed.
ERRAD	Read access denied (both ?READ and ?WRITE)
ERVBP	Invalid byte pointer passed as a system call argument
ERVWP	Invalid word pointer passed as a system call argument
ERWAD	Write access denied (?WRITE only)
ER_FS_DIRECTORY_NOT_AVAILABLE	Directory not available because the LDU was force released (AOS/VS II only)

Who Can Use It?

There are no special process privileges needed to issue this call.

For ?READ: you must have obtained a channel number to the file via ?OPEN before issuing ?READ. Also, you must have had Read access to the file at the time of the ?OPEN.

For ?WRITE: you must have obtained a channel number to the file via ?OPEN before issuing ?WRITE. Also, you must have had Write access to the file at the time of the ?OPEN.

What It Does

?READ and ?WRITE perform record I/O on an open file.

As Figure 2–200 shows, the basic ?READ/?WRITE packet has the same structure as the ?OPEN and ?CLOSE packets. Some specifications apply exclusively to ?READ and ?WRITE, while others apply to all record I/O system calls. When you issue ?READ or ?WRITE against a file, you can default some specifications to the values that you selected when you created or opened the file.

CAUTION: If you are using a pixel–mapped system, do not attempt to use device code 10 (TTI or system console input, such as a keyboard) or device code 11 (TTO or system console output, such as a screen or printer), once your operating system is running.

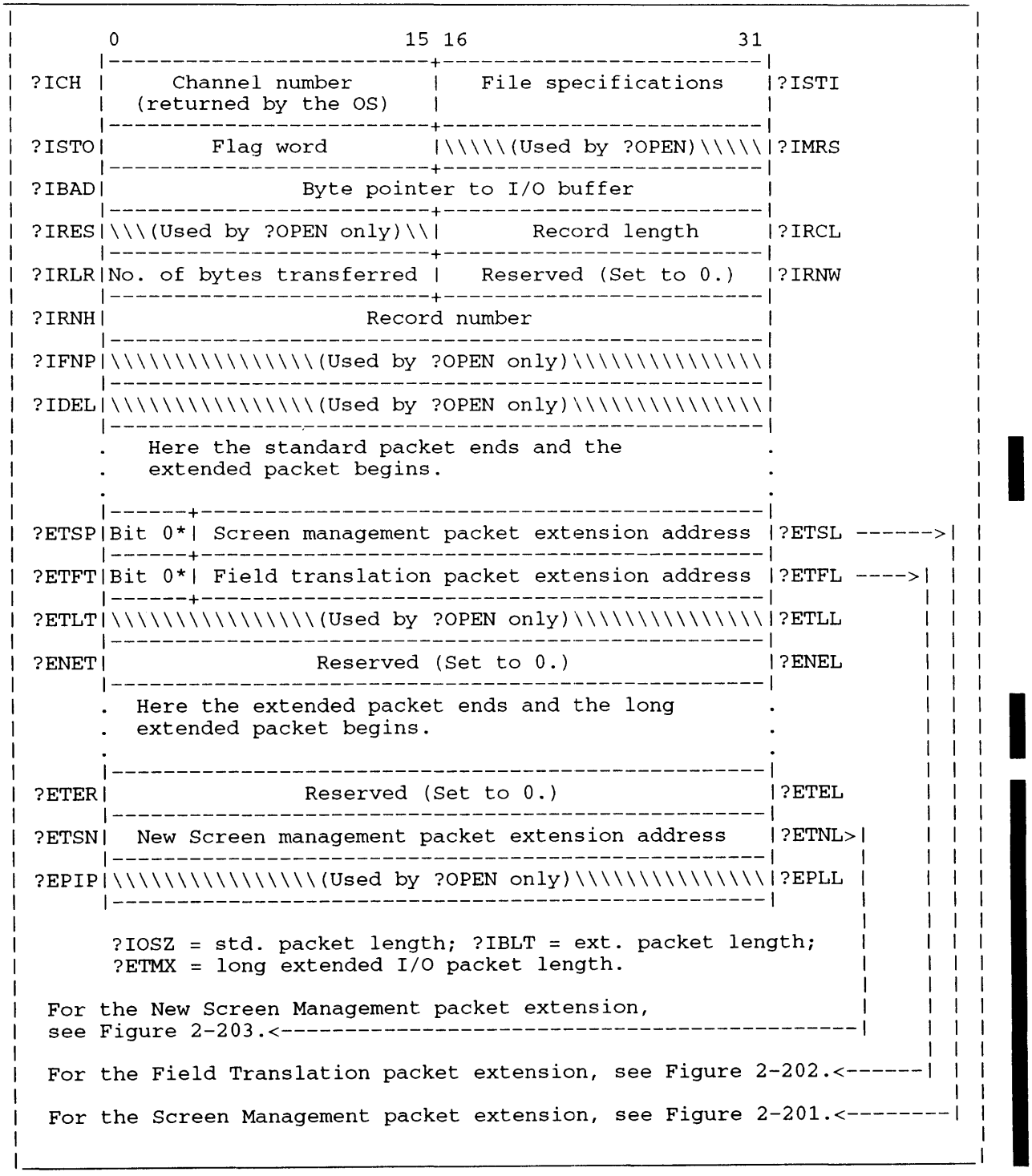


Figure 2-200. Structure of ?READ/?WRITE Packet

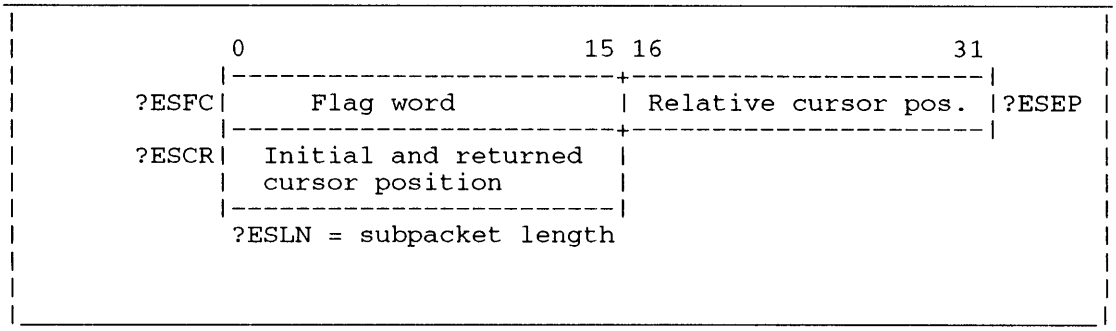


Figure 2–201. Structure of Screen Management Packet Extension

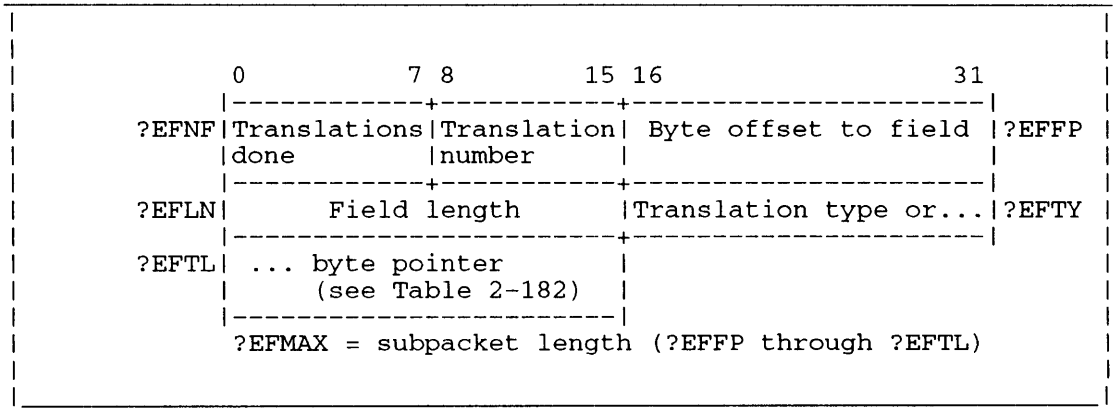


Figure 2–202. Structure of Selected Field Translation Packet Extension

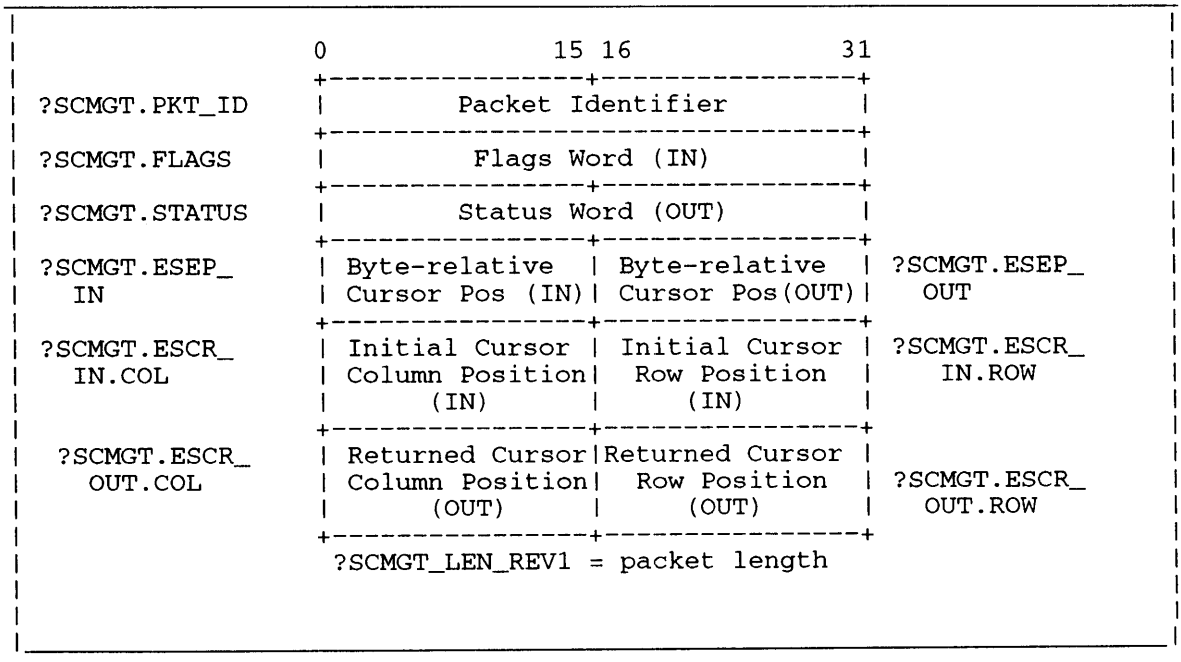


Figure 2–203. Structure of New Screen Management Packet Extension

Similarly, you can use the ?READ and ?WRITE packets to set parameters you deferred when you created or opened the file. Figure 2–200 shows the structure of the ?READ/?WRITE packet. Table 2–180 describes each offset and parameter. The structure of the Screen Management packet extension is shown in Figure 2–201, while that of the Selected field extension packet is shown in Figure 2–202. Symbols and parameters for these packets are in PARU.32.SR.

The New Screen Management packet structure is shown in Figure 2–203, and its offsets and parameters are in Table 2–183. Note that the symbols for the New Screen Management packet are in PARU_LONG.SR.

Packet Versions

There are three versions of the ?READ/?WRITE parameter packet as follows. Look at Figure 2–200 as you read the following summaries.

Standard	Includes offsets ?ICH through ?IDEL inclusive, and is ?IOSZ words long.
Extended	Includes offsets ?ICH through ?ENET inclusive, and is ?IBLT words long. Set bit ?IPKL in offset ?ISTI to indicate the extended packet.
Long Extended	Includes offsets ?ICH through ?EPIP inclusive, and is ?ETMX words long. Set bit ?IBP2 in offset ?ISTO to indicate the long extended packet.

?ISTI: The File Specifications Word

Offset ?ISTI, the file specifications word, provides a number of options for the file you are reading or writing. To default an ?ISTI field, you must explicitly set that field to 0.

The ?ISTI specifications are bit masks. Therefore, if you want to select more than one ?ISTI specification, you must OR the bit masks. For example, the specification ?ICRF!?RTDS sets the record format (?ICRF) to data–sensitive (?RTDS).

To change the file's record format, set offset ?ICRF and one of the following masks in the record format field:

- ?RTDY (dynamic–length; acts like fixed–length for terminal I/O).
- ?RTDS (data–sensitive).
- ?RTFX (fixed–length).
- ?RTVR (variable–length; acts like fixed–length for terminal I/O).
- ?RTUN (undefined–length).

  Use ?RTUN only if you are reading an unlabeled magnetic tape with an unknown block size, or if you are reading an IBM labeled tape with the undefined–length record format. The use of ?RTUN on any other type of file causes unpredictable results.

- ?RTVB (variable–block, variable–record).

  Use ?RTVB only if you are reading an IBM labeled tape with the variable block (spanned) record type.

The record format stays the same until you change it or close the file. You can also default this parameter, but only if you specified a record format when you created or opened the file. If a file has no record format parameter, the I/O system call fails, and the operating system returns an error to AC0.

Table 2–180. Contents of the Standard ?READ/?WRITE Packet*

Offset	Contents
?ICH	Channel number (returned by the OS).
?ISTI	File specifications. Packet type. ?IPKL--Indicates presence of an extended packet. DEFAULT = 0 (standard ?READ/?WRITE packet). Format select. ?ICRF--Change format to that specified in the record format field of this packet. DEFAULT = 0 (record format specified when the file was opened). File pointer positioning. ?IPST--Offset ?IRNH contains an absolute record number for fixed records and an absolute byte offset for other record types. DEFAULT = 0 (offset ?IRNH contains a relative record offset for fixed records and a relative byte offset for other record types). Append (used only by ?OPEN). Binary I/O. ?IBIN--Character device I/O is in binary mode. Binary mode is selected on each ?READ or ?WRITE request. This mode prevents the operating system from interpreting special characters and echoing the input data. DEFAULT = 0 (character device I/O is in text mode).

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

Table 2–180. Contents of the Standard ?READ/?WRITE Packet*

Offset	Contents
?ISTI (continued)	Force output. ?IFOP nonterminal I/O -- for ?WRITE, write buffer from system area. terminal I/O -- for ?READ, do not echo input; for ?WRITE, cancel outstanding Ctrl-O. DEFAULT = 0 (normal system buffering and terminal I/O).
	Exclusive open (used only by ?OPEN).
	Priority reads (used only by ?OPEN). Creation option (used only by ?OPEN).
	Input/output (used only by ?OPEN).
	Record format. ?RTDY--Dynamic length. ?RTDS--Data sensitive. ?RTFX--Fixed length. ?RTVR--Variable length. ?RTUN--Undefined length. ?RTVB--Variable block, variable record (IBM labeled tapes only).
?ISTO	Flag word. ?IMP2--Indicates a long extended packet. ?IMHN--Hold nonpriority reads.
?IMRS	Physical block size (used only by ?OPEN).
?IBAD (doubleword)	Byte pointer to I/O buffer. DEFAULT = -1 (use the I/O buffer specified when the file was opened).
?IRES	Density mode (used only by ?OPEN).
?IRCL	Record length. States the number of bytes to read or write for dynamic-length and fixed-length records and the maximum number of bytes to read or write for data-sensitive and variable-length records. DEFAULT = -1 (use the record length specified when the file was opened.) If you specified ?IRCL = -1 in the ?OPEN call, you must specify an actual record length here.

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

Table 2–180. Contents of the Standard ?READ/?WRITE Packet*

Offset	Contents
?IRLR	Actual number of bytes transferred by this I/O call (returned by the operating system).
?IRNW	Reserved. (Set to 0.)
?IRNH (doubleword)	Record number. A double-precision integer that indicates the current record number. To position the file pointer at beginning of file, set ?IPST and place 0 in ?IRNH. To position the file pointer at end of file, set ?IPST and place -1 in ?IRNH. To position the file pointer at the next sequential record, do not set ?IPST and place 0 in ?IRNH.
?IFNP (doubleword)	Byte pointer to pathname (used only by ?OPEN).
?IDEL (doubleword)	Address of delimiter table (used only by ?OPEN).

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

Positioning the File Pointer

The file–pointer position parameter, ?IPST, determines how the operating system interprets the ?IRNH offset. ?IRNH contains a double–precision record number. These two parameters let you move the file pointer forward or backward. Normally, the file pointer points to the next record before each read or write. The file pointer parameters work together as follows:

- If you set ?IPST and the record format for the file is fixed length (?RTFX), the operating system interprets the value in ?IRNH as an absolute record number (from the beginning of the file).
- If you set ?IPST and the record format is dynamic length (?RTDY), data sensitive (?RTDS), variable length (?RTVR), variable block, variable record (?RTVB), or undefined length (?RTUN), the operating system interprets the value in ?IRNH as an absolute byte count (from the beginning of the file).
- If you set ?IPST to 0 and the record format is fixed length, the operating system interprets the value in ?IRNH as a record offset, relative to the file pointer's current position; the operating system begins the I/O transfer at this position in the file.
- If you set ?IPST to 0 and the record format is other than fixed length, the operating system interprets ?IRNH as a byte offset, relative to the file pointer's current position; the operating system begins the I/O transfer at this position in the file.

You can also use ?IPST and ?IRNH to move the file pointer to either the beginning or the end of the file. An immediately following write operation overwrites data in the file at this location. To move the pointer to the beginning of the file, set ?IPST and specify 0 in ?IRNH. To move the pointer to the end of the file, set ?IPST and specify –1 in ?IRNH.

To simply read or write the next sequential record in the file (relative to the file pointer's current position), set both ?IPST and the ?IRNH pair to 0.

The binary I/O parameter, ?IBIN, applies only to character devices, and directs the operating system to pass each byte from the device as is; that is, without interpreting it. When you read from a character device in binary mode, the operating system ignores rubout characters and all control sequences. The alternative to binary mode is text mode, in which the operating system interprets each byte based on the character device's characteristics. (Text mode is the default.) To switch from binary mode input to text mode input, do one of the following:

- On your terminal, hold down the CMD key, and then press the Break key. This works only if you have allowed breaking of binary mode by setting the break function characteristic to ?CBBM or by giving the CLI command CHARACTERISTICS with the /BREAK=BMOB switch and value. (However, pressing the Break key on the system console will put you into the System Control Processor.)
- Issue a text mode ?READ.

Note that none of these actions affects binary mode output on a ?WRITE.

For terminal devices, the force output option, ?IFOP, negates the Ctrl–O character you typed previously and writes the records to the terminal. For disk files, magnetic tape files, and MCA and IPC files, this option forces the operating system to write the current buffer from a dynamic system area to the file.

If you issue ?READ/?WRITE against a pipe file, the operating system ignores the following bits of offset ?ISTI in the packet:

- ?IPST and offset ?IRNH — relative or absolute file positioning.
- ?IBIN — setting binary I/O on the pipe.
- ?IIPC — setting "IPC No Wait" on the pipe.

Screen Management Primitives

In addition to the standard packet, the operating system recognizes three packet extensions for ?READ and ?WRITE: a screen management extension, a selected field translation extension, and a new screen management interface packet extension. The new interface provides all the functionality of the previous screen management interface, as well as additional features.

You can access certain screen management primitives by appending the screen management extension to the standard ?READ or ?WRITE packet. These screen management primitives are normally controlled by the operating system's terminal services. For either ?READ or ?WRITE, this feature allows you to specify that the cursor position be returned. For ?READ, this feature lets you display an initial string, ignore typed–ahead characters, specify no echo, and enable screen–edit control characters.

Screen–edit control characters provide you with such text editing capabilities as character insertion and character overwrite, as well as the ability to position your cursor anywhere in the current buffer. Note, however, that the ability to delete characters is not a feature unique to screen edit. Also, you cannot use screen–edit control characters on a hard–copy terminal.

With both ?READ and ?WRITE, you can use the screen management extension to suppress echoing of delimiters and to specify an initial cursor position. The initial cursor position, which is expressed

in column and row–number format, places the cursor at a specific position in the display before the I/O sequence occurs. If the column or row number is too large (i.e., it exceeds the characters–per–line or lines–per–page value in characteristic word 2), the operating system reduces it to the respective limit.

To use the screen–management extension, structure the ?READ and ?WRITE packet as follows:

- Set bit ?IPKL in offset ?ISTI of the I/O packet.
- Set Bits 1 through 31 of offset ?ETSP to the address of the screen–management extension.
- Set Bit 0 of ?ETSP to 1 (to indicate the presence of the extension).

Set up the screen–management extension elsewhere in your code.

The first time you use the screen–management extended packet, set Bit 0 of offset ?ETSP to 1, so that the operating system will recognize Bits 1 through 31 in offset ?ETSP as the address of the packet extension.

After the operating system executes ?READ or ?WRITE, it clears Bit 0 of ?ETSP. If you want to change the packet extension and reuse it, reset Bit 0 of ?ETSP to 1. Otherwise, the operating system assumes that the packet has not changed and, therefore, does not examine it again. The operating system remembers the options and uses the address for the returned information.

Figure 2–204 shows the structure of the screen–management packet extension. Table 2–181 describes each offset.

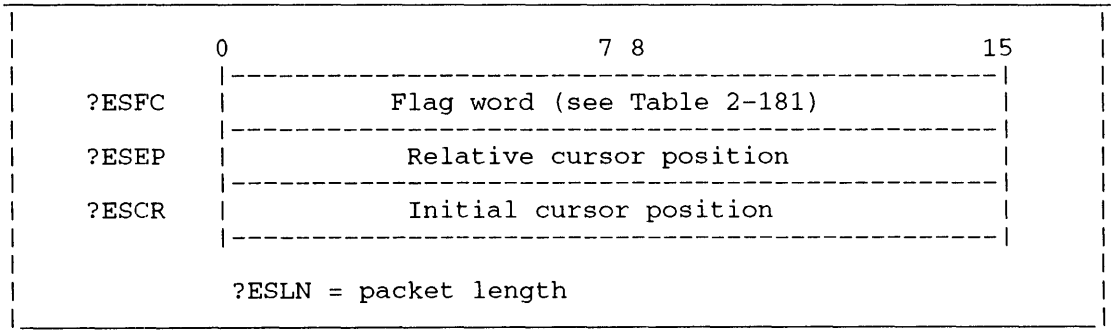


Figure 2–204. Structure of Screen–Management Packet Extension

The screen–edit control character functionality requires you to set up your input buffer with an initial character string whose final character is a valid delimiter. Without such a character string a ?READ issued to the input buffer will return error ERLTL, "Line too long".

You can also use screen–management primitives to enable autoterminating ?READ system calls. An autoterminating ?READ does not require you to enter a delimiter before the system call completes. Instead, it terminates when the user types enough characters to fill the input buffer. You can use an autoterminating ?READ to protect data that a previous ?READ or ?WRITE displayed on your screen.

Table 2–181. Contents of Screen–Management Packet Extension*

Offset	Contents
?ESFC	Flag word, with one or more of the following bit masks: ?ESSE--Screen edit; enable screen-edit control characters (CRT devices and ?READ only). If you set ?ESSE and ?IBIN in offset ?ISTI of the parameter packet, this is an attempt to specify both a screen-edit and a binary read. In this case only the binary read specification takes effect. In other words, ?IBIN in offset ?ISTI overrides ?ESSE in offset ?ESFC. ?ESRD--Redisplay data that was read previously (?READ only). (Ignored unless ?ESSE is also set.) ?ESNR--Drop typed-ahead characters; that is, flush the ring buffer before beginning input (?READ only). ?ESED--Do not echo delimiters (?READ only). ?ESCP--Initial cursor position is specified in offset ?ESCR of the packet extension. The cursor will be positioned before ?READ processing begins. Word ?ESCR of the screen-edit extended packet contains the initial cursor column (left byte) and row (right byte). If ?ESCP is not selected, then the cursor remains in the current position on the screen. ?ESDD--?READ ended with a function key; the system sets this bit (?READ only).

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

Table 2–181. Contents of Screen Management Packet Extension*

Offset	Contents
	?ESRP--Return cursor position after I/O. Upon ?READ termination, the current cursor position is returned to the user in word ?ESCR of the screen-edit extended packet (left byte, current column; right byte, current row). If option ?ESCP is also selected, then the row returned is the physical row on the screen (row numbering is zero-relative). Otherwise, it is the row relative to the starting row. This value is also zero-relative. The column position returned is always relative to the current row. ?ESNE--Do not echo input (?READ only). ?ESGT--Read typed-ahead characters, end ?READ if ring buffer is empty (error ERISO if ?ESSE is also set; ?READ only). ?ESBE--?READ ended because ring buffer is empty; the system sets this bit (?READ only). ?ESPE--The read terminated with a pointer event (?READ only; ?ESSE must also be set). ?ESBB--No echo on predisplay. (Ignored unless ?ESRD and ?ESSE are also set.)
?ESEP	Relative position within the data to place the cursor before input, where the first character position is 0. (?READ only, valid only if both ?ESSE and ?ESRD are set.)
?ESCR	Initial and returned cursor position, where the left byte of ?ESCR indicates the relative column and the right byte of ?ESCR indicates the relative row; the leftmost column, top row of the terminal is Column 0, Row 0. (This feature will not work correctly if you have not set the correct device characteristics for the terminal.)

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

NOTES: For ?READ to delimit on pointer events and thus receive flag ?ESPE, you should previously have set the ?TRPE characteristic via system call ?SECHR.

For ?READ to delimit with a function key and thus receive flag ?ESDD, you should previously have set the ?CFKT characteristic via system call ?SCHR.

Terminating a terminal ?READ without an autoterminator depends on the type of record as follows. You must have previously specified the record type in offset ?ISTI of the ?OPEN or ?READ packet.

- Fixed–length records: the user types exactly ?IRCL characters.
- Data–sensitive records: the user types a string of characters followed by a valid delimiter character. If the user types ?IRCL characters without a delimiter, the operating system terminates the read with error code ERLTL.

To issue a terminal ?READ with the autoterminating option, perform the following steps.

1. Specify the screen–edit control character functionality with ?ESSE.
2. When you set up the initial character string in the input buffer, terminate it with enough ASCII <177> characters for the longest possible delimiter. Remember that pointer device events are 16. bytes long, function key events are 2 bytes long, and standard delimiters are 1 byte long. If neither pointer device events nor function keys are delimiters, it is acceptable to use one or two ASCII <177> characters. Offset ?IBAD points to the input buffer.
3. Make sure that you specify a record length (offset ?IRCL) at least as long as the initial character string — *including* the ASCII <177> characters.

If the user types some characters and a delimiter, ?READ terminates normally. The input buffer will contain the characters the user typed along with the delimiter.

In the next three paragraphs (which refer to cursor positioning), TO means that the cursor is over the last character that differs from <177>. PAST means either that the cursor is one position after the last character that differs from <177> or else that the cursor is over the first <177> character.

If the user positions the cursor TO the last position of the field (either by typing enough characters or by using the screen–edit control characters), and then types another character, ?READ autoterminates. The input buffer will contain the characters typed followed by the Delete character.

If the user positions the cursor PAST the last position of the field (by using the screen–edit control characters), and then types another character, ?READ autoterminates. The additional character will overwrite the first Delete character in the input buffer.

If the user positions the cursor PAST the last position of the field (by using the screen–edit control characters), edits the line without typing over the last position of the field, and then types a delimiter, ?READ terminates normally. As with any screen–edit read with the ?ESSE option, the delimiter will be at the end of the line regardless of the cursor position when the user typed the delimiter. The input buffer will contain the entire line, with the delimiter having overwritten the first Delete character. However, if the delimiter is a function key, it overwrites both Delete characters.

You can also use screen management primitives to perform self–terminating ?READ functions. A self–terminating ?READ terminates itself after reading all the characters in the input buffer. It will not terminate, however, until at least one character has been read from the buffer. This type of ?READ is most useful if your application program can handle data as soon as it is entered, without waiting for a delimiter or a specific number of characters to be typed. It can be used as an alternative to single–byte I/O, or in an environment where data of varying amounts tends to arrive in spurts.

In order to perform a self–terminating ?READ you must do the following:

1. Supply a screen–management extension to the ?READ system call.
2. Set bit ?ESGT in the screen management packet.
3. If there is no data in the input ring buffer, the ?READ system call waits for a character to be entered.
4. If there is data in the input ring buffer, the ?READ system call takes characters from the input ring buffer until it is empty, or until the condition to terminate the ?READ (entering a delimiter, for a data–sensitive ?READ or entering ?IRCL characters, for a fixed–length ?READ) is met.

5. If The ?READ terminates because there are no more characters in the input ring buffer, the system sets bit ?ESBE in the screen management packet extension.

Restriction: If you set bit ?ESGT in the screen management packet, you must NOT set bit ?ESSE.

Selected Field Translation

The selected field translation extension lets you translate one or more fields within a record during the I/O sequence. The translations this packet allows are ASCII lowercase to ASCII uppercase or vice versa, and ASCII to EBCDIC or vice versa. In addition, you can select one of the following three parity options for the record:

- Even parity
- Odd parity
- No parity (strip parity bits)

The procedure for using the selected field translation extension is similar to the procedure for the screen management extension. Briefly, you must perform the following steps:

1. Set bit ?IPKL in offset ?ISTI of the standard ?READ or ?WRITE packet.
2. Set Bits 1 through 31 of offsets ?ETFT to the address of the field translation packet.
3. Set Bit 0 of ?ETFT to 1 (to indicate the presence of the extension).

Supply the packet elsewhere in your program.

To reserve the correct number of words for this extension, you must begin it or end it with the pseudo–op

```
.LOC  extended_pkt_address+?EFNF+n
```

where

n	is the number of the subpacket * subpacket length (See the following for information on subpackets.).

The first time you use the selected field translation extended packet, set Bit 0 of offset ?ETFT to 1 (See Figure 2–200 for ?ETFT packet location), so that the operating system will recognize Bits 1 through 31 in offset ?ETFT as the address of the field translation packet extension.

After the operating sytem executes ?READ or ?WRITE, it clears Bit 0 of ?ETFT. If you want to change the packet extension and reuse it, reset Bit 0 of ?ETFT to 1. Otherwise, the operating system assumes that the packet has not changed and, therefore, does not examine it again. The operating system remembers the options and uses the address for the returned information.

Figure 2–205 shows the structure of the packet extension for selected field translation, and Table 2–182 describes the contents of each offset.

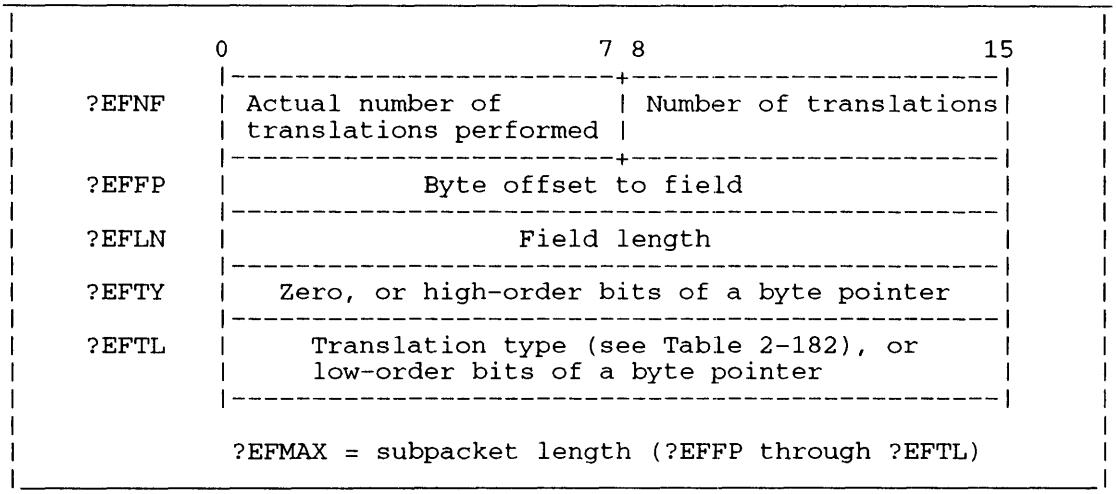


Figure 2-205. Structure of Selected Field Translation Packet Extension

Offsets ?EFFP, ?EFLN, ?EFTY, and ?EFTL form a subpacket within the field translation packet. You must use one subpacket for each translation you want to perform. Use the following pseudo–op after each subpacket to space past that subpacket:

```
.LOC  subpacket_address+?EFMAX
```

Table 2–182. Contents of Selected Field Translation Packet Extension*

Offset	Contents
?EFNF	Right byte: Number of translations to perform. Left byte: Actual number of translations performed. (The OS returns this value.)
	?EFFP Byte offset to field (within the specified record). ?EFLN Field length (in bytes). Default = 0 (translates actual number of bytes transferred)
	?EFTY/ ?EFTL Translation type.
Translation subpacket: one per translation	?ASEB--ASCII to EBCDIC. ?EBAS--EBCDIC to ASCII. ?ALAU--ASCII lowercase to ASCII uppercase. ?AUAL--ASCII uppercase to ASCII lowercase. ?SPAR--Strip parity bits. ?AEPR--Add even parity. ?AOPR--Add odd parity.
	NOTE: Instead of these masks, you can use offsets ?EFTY and ?EFTL as a byte pointer to a 256-byte translation table.

* There is no default unless otherwise specified.

The left byte of offset ?EFNF specifies the number of translations for this ?READ or ?WRITE and, therefore, the number of subpackets.

The operating system performs the translations in the order in which the subpackets appear. Thus, if you want to translate a field from EBCDIC to ASCII, and then add odd parity, specify the EBCDIC to ASCII translation in the first subpacket, and specify odd parity in the second subpacket.

Note that the translation tables work in only one direction. For example, if you want to translate an EBCDIC field to ASCII on a ?READ, and then write the records back in EBCDIC format, you must redefine the translation type before performing the ?WRITE.

The program excerpt in Figure 2–206 shows two translations on the same record.

```
PKT:    .LOC    PKT+?EFNF          ;Right byte: No. of translations?
        .WORD   2                  ;Two.

        .LOC    PKT+?EFNF+2*?EFMAX ;Reserve space for two
                                   ;   subpackets.

;"TRNS0" is the first subpacket (for first field translation).

TRNS0: .LOC     TRNS0+?EFFP        ;Byte offset to field.
        .WORD   0                  ;Field starts with first byte.

        .LOC    TRNS0+?EFLN        ;Field length.
        .WORD   8.                 ;Eight bytes (four 16-bit words).

        .LOC    TRNS0+?EFTY        ;Translation type
        .DWORD  ?ALAU              ;ASCII lowercase to uppercase.

        .LOC    TRNS0+?EFMAX       ;Space past first subpacket.

; "TRNS1" is the second subpacket (for second field translation).

TRNS1: .LOC     TRNS1+?EFFP        ;Byte offset to field.
        .WORD   8.                 ;Field starts with ninth byte.

        .LOC    TRNS1+?EFLN        ;Field length.
        .WORD   0                  ;Default (rest of record)

        .LOC    TRNS1+?EFTY        ;Translation type
        .DWORD  ?ALAU              ;ASCII lowercase to uppercase.

        .LOC    TRNS1+?EFMAX       ;Space past second subpacket.
```

Figure 2–206. Selected Field Translation Packet Sample Listing

Sample Packet

```
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!?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
        .WORD   33.             ;?WRITE use this value.) Maximum
                                ;record length is 33-byte fixed
                                ;records.
        .LOC    PKT+?IRLR       ;Record length.  (?READ and ?WRITE
        .WORD   0               ;use this value.)  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.
```

New Screen Management Interface

There is a new Screen Management Interface you can optionally use to replace the existing screen management interface. Many of the features of the new screen management interface are identical to those of the old screen management interface. Additional features of the new packet are:

- Getting typed ahead characters (unpended) using ?SCMGT.FLAGS.ESUG.
- Starting a read in insert mode using ?SCMGT.FLAGS.ESIM.
- Returning the relative cursor position using ?SCMGT.FLAGS.ESCO.
- Returning the date on timeout using ?SCMGT.FLAGS.ESRT.
- Autoterminating a read with the left arrow key using ?SCMGT.FLAGS.ESLT.
- Defining a normal carriage return using ?SCMGT.FLAGS.ESNC.
- Using nulls as spaces with ?SCMGT.FLAGS.ESNP.
- Returning status flags using ?SCMGT.STATUS.
- Reading was ended in Insert Mode using ?SCMGT.STATUS.ESII.
- Verifying that the read buffer was modified using ?SCMGT.STATUS.ESMI.
- Defining the initial cursor column and row positions using ?SCMGT.ESCR_IN.COL and ?SCMGT.ESCR_IN.ROW respectively.
- Returning cursor column and row positions using ?SCMGT.ESCR_OUT.COL and ?SCMGT.ESCR_OUT.ROWrespectively.

All New Screen Management packet offsets are supported in AOS/VS II; these packet offsets and parameters are shown in Table 2–183, and the flag words are in Table 2–184. Not all of the new screen–management features described in Table 2–183 are supported in AOS/VS, and Table 2–185 details the current exceptions.

Where the old screen management interface uses the ?READ/?WRITE extended packet offset ?ETSP, the new interface uses the ?READ/?WRITE long extended packet offset ?ETSN to designate the New Screen Management packet. Because the new interface is in the long extended packet, you must also set ?ISTO flag ?IMP2 to use the packet.

NOTE: ?SCMGT.FLAGS bits 5,9, and 10 are reserved so that flags ?ESSE to ?ESBB are compatible with the old screen management packet interface. ?SCMGT.STATUS bits 0 through 4 and 6 through 8 are also reserved, so that flags ?ESDD to ?ESPE are compatible with the old packet interface. The content description for each offset in Table 2–183 includes new offsets identical in function to the old packet interface offsets. The old screen management interface symbol for identical bits is listed after each bit description.

When setting the cursor position with ?SCMGT.FLAGS.ESCP, provide either the cursor's row and column positions in ?SCMGT.ESCR_IN, or the cursor's relative position in ?SCMGT.ESEP_IN.

The unpended read returns control to your program, with a character count of zero, even if no characters are present in the input ring buffer. The pended read flag is ?SCMGT.FLAGS.ESGT; the unpended version is ?SCMGT.FLAGS.ESUG.

The left–arrow autoterm read, offset ?SCMGT.FLAGS.ESLT, is terminated if the left arrow key is pressed before input into the buffer.

You can start in insert mode (the same as typing a control–E at the CLI prompt to insert information into a line) when you use the flag ?SCMGT.FLAGS.ESIM.

A carriage return is treated as a delimiter without line truncation, when you use the flag ?SCMGT.FLAGS.ESNC.

Use the flag ?SCMGT.FLAGS.ESNP to treat <000> as a placeholder in the buffer. These nulls will be treated as spaces. If you use this option, you should not use null as a delimiter.

To use the New Screen Management extension, structure the ?READ and ?WRITE packet as follows:

- Set bit ?IBP2 in offset ?ISTO of the I/O packet.
- Set offset ?ETSN to the address of the New ScreenManagement extension.

Set up the New Screen Management extension elsewhere in your code.

Table 2–183. Contents of the New Screen Management Packet.

Offset	Contents
?SCMGT.PKT_ID (double word)	Packet identifier; set to ?SCMGT_PKTID_REV1.
?SCMGT.FLAGS (double word)	Flag word. Use these bits to select read options (input only). The flag word bit positions are numbered. Unused bits are reserved and must be set to zero. ?SCMGT.FLAGS.ESSE = 0. Set this flag to enable Screen Edit. Only valid on CRT devices; error if hardcopy device. Error if ESGT flag is set. Error if ESUG flag is set. Ignored for binary read requests. Same as ?ESSE. ?SCMGT.FLAGS.ESRD = 1. Set this flag to predisplay the ?IBAD buffer at the start of the read. Ignored if ESSE flag is not set. Ignored if ESBB flag is set. Same as ?ESRD. ?SCMGT.FLAGS.ESNR = 2. Set this flag to flush the input ring buffer before getting any input characters. Same as ?ESNR. ?SCMGT.FLAGS.ESED = 3. Set this flag to stop echoing delimiters. Same as ?ESED. ?SCMGT.FLAGS.ESCP = 4. Before starting to read, set this flag to move the cursor to the position specified in offsets ?SCMGT.ESCR_IN.COL and ?SCMGT.ESCR_IN.ROW. Same as ?ESCP. ?SCMGT.FLAGS.ESDD = 5. Reserved for the old screen management packet. Same as ?ESDD. ?SCMGT.FLAGS.ESRP = 6. Set this flag to return the cursor position in offsets ?SCMGT.ESCR_OUT.COL and ?SCMGT.ESCR_OUT.ROW at the end of the read. Same as ?ESRP. ?SCMGT.FLAGS.ESNE = 7. Set this flag to suppress echoing of all characters entered for this read. Same as ?ESNE. ?SCMGT.FLAGS.ESGT = 8. Set this flag to get typed-ahead input. Error if ESSE flag is set. Error if ESUG flag is set. Same as ?ESGT.

(continued)

Table 2–183 . Contents of the New Screen Management Packet.

Offset	Contents
?SCMGT.FLAGS (double word) (continued)	?SCMGT.FLAGS.ESBB = 11. Set this flag to use the Predisplay buffer as input, but not display it on the screen; no echo on Predisplay. Ignored if ESSE flag is not set. Error if ESCP flag is not set. Same as ?ESBB. ?SCMGT.FLAGS.ESUG = 12. Set this flag to get typed-ahead input without pending if the input ring buffer is empty. Error if ESSE flag is set. Error if ESGT flag is set. ?SCMGT.FLAGS.ESIM = 13. Set this flag to start in Insert Mode. Ignored if ESSE flag is not set. ?SCMGT.FLAGS.ESCO = 14. Set this flag to return the relative cursor position (in bytes) from the start of the read buffer in offset ?SCMGT.ESEP_OUT. Ignored if ESSE flag is not set. ?SCMGT.FLAGS.ESRT = 15. Set this flag to return the entire input buffer, if the read times out. ?SCMGT.FLAGS.ESLT = 16. Set this flag to cause the read to terminate if the left arrow is used to move the cursor before the start of the input. Ignored if ESSE flag is not set. Ignored if neither ESRD nor ESBB are set. Ignored if the read buffer is not initialized for auto-termination. ?SCMGT.FLAGS.ESNC = 17. Set this flag to suppress the special behavior of Carriage Return for screen-edit reads. Ignored if ESSE flag is not set. ?SCMGT.FLAGS.ESNP = 18. Set this flag to treat nulls as spaces. Ignored if ESSE flag is not set. Ignored if <000> is a delimiter.

(concluded)

The New Screen Management Status Word Packet offsets are defined in Table 2–184.

Table 2–184. Contents of the New Screen Management Status Word Packet.

Offset	Contents
?SCMGT.STATUS (double word)	This status word's bits return information to the caller. Bit positions are numbered. Unused bits are reserved and should be ignored. Some bit positions are reserved to provide easy conversion between the old and new packets. ?SCMGT.STATUS.ESDD = 5. The system sets this flag, if the read ended with a function key delimiter. Same as ?ESDD. ?SCMGT.STATUS.ESBE = 9. The system sets this flag, if a get type-ahead read terminated from an empty input buffer, rather than a delimiter or maximum number number of characters. Same as ?ESBE. ?SCMGT.STATUS.ESPE = 10. The system sets this flag, if the read ended with a mouse event delimiter. Same as ?ESPE. ?SCMGT.STATUS.ESII = 11. The system sets this flag, if the read ended in Insert Mode. ?SCMGT.STATUS.ESMI = 12. The system sets this flag if any characters were entered into the user's buffer during a Screen Edit read.
?SCMGT.ESEP_IN (word)	At the start of the read, this offset contains the desired relative cursor position within the Predisplay buffer, as a byte offset from the beginning of the read buffer. Ignored unless both ESSE and ESRD flags are set.
?SCMGT.ESEP_OUT (word)	This offset returns the relative cursor position as a byte offset from the beginning of the read buffer. Ignored unless both ESSE and ESCO flags are set.
?SCMGT.ESCR_IN.COL (word)	This offset contains the initial cursor column position. Ignored unless ESCP flag is set.
?SCMGT.ESCR_IN.ROW (word)	This offset contains the initial cursor row position. Ignored unless ESCP flag is set.
?SCMGT.ESCR_OUT.COL (word)	This offset contains the initial cursor column position. Ignored unless ESRP flag is set.
?SCMGT.ESCR_OUT.ROW (word)	This offset contains the initial cursor row position. Ignored unless ESRP flag is set.

Dual Asynchronous Receiver–Transmitters (DRTs) and Intelligent Asynchronous Controllers (IACs, supporting Data Channel or Local Bus protocols) control screen management on the MV/Family host and execute on the host or on the IAC controller.

In AOS/VS, several New Screen Management packet features are unsupported on the host or on the IAC controller. Table 2–185 lists supported (Y for yes) and unsupported (N for no) new Screen–Management features for either the host or the IAC controller implementations. All of these new Screen–Management features are fully supported in AOS/VS II.

Table 2–185. Supported (Y) and Unsupported (N) New Screen Management Packet Features in AOS/VS

Feature and offset	Locus of Execution: On Host	Locus of Execution: On IAC
Left autotermination ?SCMGT.FLAGS.ESLT=16	Y	Y
Normal carriage return ?SCMGT.FLAGS.ESNC=17	Y	Y
Return entire buffer on timeout ?SCMGT.FLAGS.ESRT=15	N	Y[1]
Null as a space ?SCMGT.FLAGS.ESNP	Y	Y
Read ended in insert mode ?SCMGT.STATUS.ESII=11	Y	Y
Read buffer modified ?SCMGT.STATUS.ESMI	Y	Y
Relative cursor column returned ?CMGT.ESEP_OUT	N	Y[1]

[1] 68000-based IACs support this feature; IACs prior to 68000-based ones do not support this feature.

Notes

- See the descriptions of ?GECHR, ?OPEN, and ?CLOSE in this chapter.

?REC — Receives an intertask message.

?REC
error return
normal return

Input		Output	
AC0	Address of the mailbox that is to receive the intertask message	AC0	Unchanged
AC1	Reserved (Set to 0.)	AC1	Intertask message
AC2	Reserved (Set to 0.)	AC2	Undefined

Error Codes in AC0

No error codes are currently defined.

Why Use It?

?REC is one of several system calls for intertask communications. (The others are ?RECNW, ?XMT, ?XMTW, ?SIGWT, ?WTSIG, and ?SIGNL.) Generally, you use intertask communications to synchronize tasks and to manage critical regions (critical regions are areas of code which need to be protected from being executed by multiple tasks at a given time.)

Because you can use the intertask message facility to pass addresses, the mailboxes for 32–bit tasks must be 32 bits wide, and those for 16–bit tasks must be 16 bits wide.

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

?REC loads AC1 with a message sent by another task, and then clears the contents of the mailbox (that is, sets the mailbox contents to 0). The operating system then readies the receiving task, provided it was not suspended in the meantime by a ?PRSUS or ?IDSUS call.

If the receiving task issues ?REC before the sending task transmits the message, via ?XMTW or ?XMT, the operating system suspends the ?REC caller until the transmission occurs.

If more than one task issues ?REC against the same mailbox, the highest priority task receives the message, unless the sending task specified that the message should be broadcast to all potential receivers. If the message is not broadcast and all receiving tasks have the same priority, the operating system sends the message to the highest priority receiver and suspends the other tasks that issued subsequent ?RECs against the mailbox.

Notes

- See the descriptions of ?PRSUS, ?RECNW, ?IDSUS, ?XMTW, and ?XMT in this chapter.

?RECNW — Receives an intertask message without waiting.

?RECNW
error return
normal return

Input

AC0	Address of the mailbox that is to receive the intertask message
AC1	Reserved (Set to 0.)
AC2	Reserved (Set to 0.)

Output

AC0	Unchanged
AC1	Intertask message
AC2	Undefined

Error Codes in AC0

ERNMW No message waiting (There is no outstanding message.)

Why Use It?

Like ?REC, ?RECNW allows a task to receive a message from another task. You would choose ?RECNW over ?REC to avoid suspending the receiving task if there is a delay between the receive and the complementary transmit 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

?RECNW loads AC1 with a message sent by another task, and then clears the contents of the mailbox. The operating system then readies the receiving task, provided it was not suspended in the meantime by an ?IDSUS or a ?PRSUS call.

?RECNW, unlike ?REC, does not suspend the calling task if the mailbox is empty. Instead, ?RECNW fails and returns an error code in AC0. ?RECNW maintains the calling task in the ready state, regardless of the timing of the transmit and receive sequence. This means that a receiving task can issue ?RECNW before the sending task transmits the message (?XMT or ?XMTW) without risking suspension. If the sending task becomes suspended on an ?XMTW to the mailbox, a complementary ?RECNW causes rescheduling.

Notes

- See the descriptions of ?PRSUS, ?IDSUS, ?XMTW, and ?XMT in this chapter.

?RECREATE
error return
normal return

Input

AC0	Byte pointer to the target file's pathname
AC1	Reserved (Set to 0.)
AC2	Reserved (Set to 0.)

Output

AC0	Unchanged
AC1	Undefined
AC2	Undefined

Error Codes in AC0

ERDID	Directory delete error
ERDIU	Directory in use
ERFAD	File access denied
ERFDE	File does not exist
ERMPR	System call parameter address error
ERPRM	Cannot delete permanent file
ERVBP	Invalid byte pointer passed as a system call argument

Why Use It?

?RECREATE combines the functions of ?DELETE and ?CREATE in that it allows you to delete all user data in a file, but maintain its characteristics. ?RECREATE is useful for releasing a file's disk blocks.

Who Can Use It?

There are no special process privileges needed to issue this call. If you don't have Owner access to the file, you must have Execute, Write, and Read access to the file's parent directory. If you do have Owner access to the file, you must have Execute access to the file's parent directory and either Write or Append access to the file's parent directory.

What It Does

?RECREATE deletes the target file or directory, and creates it again with the same ACL, filename, file type, creation date, and user data area (UDA) information, but without data. In short, ?RECREATE gives you a file with the same characteristics as your original file, but with a length of 0.

If the target file is open when you issue ?RECREATE, the operating system maintains all the data until you issue the last ?CLOSE. However, the data is visible only to the processes that opened the file before you issued ?RECREATE.

You cannot issue ?RECREATE against a file or directory with the PERMANENCE attribute or a directory that is currently in use (that is, your working directory or one cited in a search list).

Notes

- See the descriptions of ?CLOSE, ?CREATE, and ?DELETE in this chapter.

?RELEASE
error return
normal return

Input

AC0	Byte pointer to the pathname of the logical disk unit (LDU)
AC1	One of the following: • Reserved in AOS/VS (Set to 0.) • ?RLFRC, in AOS/VS II, to force the release of the LDU named by the AC0 byte pointer.
AC2	Reserved (Set to 0.)

Output

AC0	Unchanged
AC1	Undefined
AC2	Undefined

Error Codes in AC0

ERIFT Illegal file type
ERVBP Invalid byte pointer passed as a system call argument
ERVIU LDU in use, cannot release (AOS/VS only)
ERWAD Write access denied
ER_FS_CANNOT_RELEASE_RESTRICTED_LDU
Logical disks containing the ROOT, PAGE, or SWAP directories cannot be force released (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?

Once a logical disk has been initialized, it remains initialized until you release it. Therefore, you must issue ?RELEASE to reuse the disk drives on which an LDU's component volumes are mounted. In AOS/VS you cannot release an LDU if it is in use.

In AOS/VS II, if you use the ?RLFC option, you can release an LDU, even if it is in use. Therefore choose this feature when you need to release an LDU, but cannot identify who has files open in the LDU.

Who Can Use It?

There are no special process privileges needed to issue this call. You must have Execute access to the logical disk's parent directory and Owner access to the logical disk itself.

What It Does

?RELEASE releases an LDU that you previously initialized with ?INIT. In AOS/VS, you cannot release an LD that is currently in use.

In AOS/VS II, you can force–release an LDU by setting the option flag ?RLFRC in AC1. The forced release of an LDU is similar to an emergency shutdown release of an LDU. Any LDU you specify in AC0 is released, except one containing the ROOT, PAGE, or SWAP directories. If you try to release any LDU containing the ROOT, PAGE, or SWAP directories, the operating system will return the error ER_FS_CANNOT_RELEASE_RESTRICTED_LDU. The operating system releases an LDU regardless of the number of open files, and ?RELEASE then logs errors about any attempt to remove a force–released LDU.

Active mirror synchronizations are broken during the LDU forced release. All open files in the target LDU are forced to close, and file attributes are updated to the disk. But any user's channel remains open until closed by the process that opened the channel.

CAUTION: If you force release an LDU, any open file, shared access file, or other user–buffered data from the file system on the LDU will not be written to disk or updated. Such files could be corrupted.

Subsequent requests to force–closed files will return the error ER_FS_DIR_NOT_AVAILABLE until the channel is closed, or until the process that opened the channel is terminated.

?RELEASE releases any LDUs subordinate to the force–released LDU. Any subsequent request to use any force–released LDU is aborted and reported to the operator console. Files are accessible again after the force–released LDU is reinitialized.

Notes

- See the description of ?INIT in this chapter.

?RENAME — Renames a file or pathname.

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

Input

AC0 One of the following:

- Byte pointer to the file's original filename
- 0 if a packet address is supplied

AC1 Byte pointer to the new filename (AOS/VS and AOS/VS II) or new pathname (AOS/VS II only)

AC2 One of the following:

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

Output

AC0 Unchanged

AC1 Unchanged

AC2 Undefined or address of ?RENAME packet if not supplying a packet

Error Codes in AC0

ERIFC Illegal filename character
ERIFT Illegal file type
ERCRR Cannot rename the root
ERCPD Control point directory maximum size exceeded
ERVBP Invalid byte pointer passed as a system call argument
ERWAD Write access denied
ER_FS_CANNOT_CROSS_LDU_BOUNDARY
Cannot rename across a logical disk unit (LDU) boundary (AOS/VS II only)
ER_FS_CANNOT_RENAME_UNDER_HIERARCHY
Cannot rename a directory within its own directory hierarchy (AOS/VS II only)
ER_FS_DIRECTORY_NOT_AVAILABLE
Directory not available because the LDU was force released (AOS/VS II only)
ER_FS_WRITE_OR_APPEND_ACCESS_REQ
Write or append access is required (AOS/VS II only)

Why Use It?

You can use ?RENAME to change the name of a file or LDU. You can also use ?RENAME to move a file or directory to a different location within the same logical disk unit's directory hierarchy. ?RENAME is much faster than moving the file because the move copies all the data blocks.

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 Owner access to the target file or Write access to the target file's parent directory. 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

In AOS/VS, and in releases prior to AOS/VS II 2.20, ?RENAME deletes the target file's current pathname (a filename including a ":", "@", "=", or a "^") or name and gives the file the new name you specify in AC1.

In AOS/VS II release 2.20 only, ?RENAME deletes the target file's current pathname (a filename including a ":", "@", "=", or a "^") or name and gives the file the new *pathname* or name you specify in AC1.

Specifying a pathname renames a file across directories, and the following restrictions apply. When you use ?RENAME across directories, the parent directory of the pathname or filename in AC1 must be within the same LDU as the target file, and the new pathname or filename cannot exist in the target directory hierarchy, or you will produce the ERNAE error.

When you use ?RENAME across directories, the calling process must have either write or append access to the target file's new parent directory (the parent directory of the new name specified in AC1).

The target filename can be specified in one of two ways: either by a byte pointer to the original pathname in AC0, or by using offset ?GCPCN in the ?RENAME packet. Only use the packet if you need to specify the target file's channel number. If you use a channel number, you must set AC0 to zero to indicate the use of a channel number. If a byte pointer is specified in AC0, then the current (original) name may be a pathname; the new name can either be a filename or a full pathname depending on your software release.

If you rename an LDU, the new name becomes invalid if you use ?RELEASE to release the LDU, and you then reinitialize it. The LDU resumes its original name.

You can only rename a restricted set of files types if you are renaming across directories. You cannot use ?RENAME to rename the following file types across directories:

Symbol	Restricted File Type
?FLDU	LDU files.
?LIPC through ?HIPC	All console and IPC files.
?LUNT through ?HUNT	All unit and device files.
?FGFN	Generic files, for example @DATA, @LIST, @NULL must reside in :PER.
?FMTF	Magnetic tape files.
?FREM	Remote host – Remote Access (RMA) files.
?FHST	Remote host – X.25 access files.
?FNPN	Network process name files.

If you rename a restricted file, the ERIFT error results.

The rename–across–directories feature requires that the new parent directory (and all directories up to the logical disk root directory) has enough space remaining to accommodate the target file and any subordinate directory hierarchy. Space is limited by the space remaining in any control point directory.

To use ?RENAME to rename files, the calling process must have Write access to the target file's parent directory, or Owner access to the target file.

Figure 2–207 shows the structure of the ?RENAME packet.

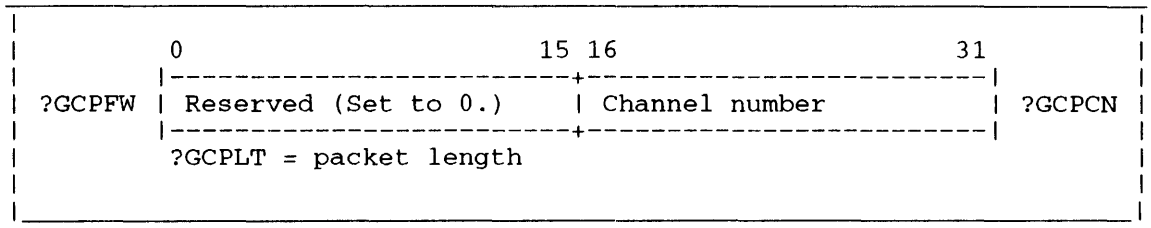


Figure 2–207. Structure of ?RENAME Packet

?RESCHED

Reschedules current time slice (32–bit processes only).

?RESCHED
error return
normal return

Operating System Differences

See the section "What It Does" below.

Input

None

Output

None

Error Codes in AC0

ERICM 16–bit program incorrectly attempted to call ?RESCHED

Why Use It?

?RESCHED can be valuable when multiple processes are cooperating. For example, if an executing process reaches a roadblock and must wait for an external event that is triggered by another of the cooperating processes, the executing process can relinquish control of the CPU. Then, the next time the operating system schedules that process, it can see if the roadblock has cleared.

A process that cannot proceed can issue ?RESCHED to allow another process to run.

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

Under AOS/VS, ?RESCHED has the calling process give up control of the CPU for the rest of the process' time slice. Under AOS/RT32, ?RESCHED has the calling process give up control of the CPU immediately since AOS/RT32 doesn't perform time slicing. In either case ?RESCHED causes the operating system to immediately schedule another process for execution.

?RESIGN — Resigns as a server.

?RESIGN
error return
normal return

Input

None

Output

None

Error Codes in AC0

ERNAS Process is not a server

Why Use It?

You can issue ?RESIGN to resign as a server process. Note that ?RESIGN alone does not clear the customer/server entry from the connection table. The operating system clears the entry only when both the customer and the server disconnect.

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 process.

What It Does

?RESIGN breaks the connection between the calling process and its customers, and sends obituary messages to the customers (unless they set the ?MCOBIT mask in their ?CON system calls). The calling process issues ?RESIGN to revoke the ?SERVE it issued previously. If the caller is not a server, ?RESIGN fails, and the operating system returns error code ERNAS to AC0.

Notes

- See the descriptions of ?SERVE, ?BRKFL, ?CTERM, ?DCON, ?RETURN, and ?TERM in this chapter.

?RETURN — Terminates the calling process and passes the termination message to the father.

?RETURN
error return

Input

AC0 Error code to the CLI (optional)

AC1 Byte pointer to the message (optional)

AC2 Contains the following:

- Bits 0 through 15 (the high–order bits) set to zero
- Bits 16 through 23 (the low–order bits) contain the following flags and data:

  ?RFCF—Message is in CLI format
  ?RFEC—Error code is in AC0
  ?RFWA—Warning (severity bit)
  ?RFER—Error (severity bit)
  ?RFAB—Abort (severity bit)

- Bits 24 through 31 contain the byte length of the message (message length must be between 0 and 255 characters)

Output

None (There is no normal return from ?RETURN.)

Error Codes in AC0

ERMPR	System call parameter address error
ERVBP	Invalid byte pointer passed as a system call argument

Why Use It?

?RETURN lets you terminate the calling process and pass control back to its father process with an optional termination message from the son. If the father process is the CLI, set the severity bits in AC2. Setting the severity bits in AC2 causes the CLI to flag the reason for the process termination.

Who Can Use It?

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

?RETURN Continued

What It Does

?RETURN terminates the calling process and returns control to its father.

To send a termination message to the caller's father, load AC1 with a byte pointer to the message, and load Bits 24 through 31 of AC2 with the byte length of the message.

Flag ?RFCF signals that the message is in CLI format. You should set ?RFCF if you intend to send a message, if the father process is EXEC, or if the father process is the CLI. If the father is neither EXEC nor the CLI, you can send the message in any format that the calling process and its father agree upon.

If there is to be no message, set Bits 24 through 31 of AC2 to 0. This directs the operating system to ignore the contents of AC1 and to send a termination code to the father via the IPC facility.

?RETURN also generates an error code in AC0. The severity bits in AC2 direct the father process to handle this error as one of the following types: a warning (?RFWA), an error (?RFER), or an abort condition (?RFAB). If you set one of these bits and the father process is the CLI, the CLI displays

- The message that you specified in AC1 (optional).
- The system error message that corresponds to the error code in AC0, flagged as a WARNING, ERROR, or ABORT.

Note that ?RETURN does not actually affect how the CLI or EXEC handles warnings, errors, and aborts.

To flag a normal return, set AC2 to 0 or specify ?RFCF in AC2.

If a ?RETURN breaks a customer/server connection, the operating system sends both an obituary message and a standard termination message to the father process. Thus, if a server is also the father of its customer, it must issue two ?IREC system calls to receive both messages when the customer issues ?RETURN.

?RINGLD — Loads a program file into a specified ring.

?RINGLD
error return
normal return

Input

AC0	Byte pointer to the program file's pathname
AC1	Reserved (Set to 0.)
AC2	Reserved (Set to 0.)

Output

AC0	Unchanged
AC1	Undefined
AC2	Undefined

Error Codes in AC0

EREAD	Execute access denied
ERICM	Illegal system command (invalid call for 16–bit processes)
ERMEM	Insufficient memory available
ERPDF	Error in process UST definition
ERPTY	Illegal process type
ERRAL	Ring already loaded
ERRNI	Ring number invalid
ERRTB	Ring too big (The specified program file is too large for the ring.)
ERVBP	Invalid byte pointer passed as a system call argument

Why Use It?

?RINGLD lets you define code in another user ring. You can use ?RINGLD to load runtime routines into another ring for shared access by processes.

Who Can Use It?

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

What It Does

?RINGLD loads the program file (.PR file) that you specify in AC0 into Ring 4, 5, or 6. You can issue ?RINGLD from any user rings, including Ring 7, to load another user ring.

The program that you load must contain a properly defined gate array. The gate array identifies the legal entry points for the ring you are loading. In addition, you must access one of the legal gates in the calling process's program.

You can address a program in an inner ring by issuing LCALL instructions (subroutine calls) from tasks within the calling process.

When you link a program that you intend to load into an inner ring, you should set its task count greater than or equal to the task maximum that you set for the calling process. If the new

program's task count is greater than that of the calling process, the operating system readjusts it at ?RINGLD time to match the calling process's maximum task count.

The operating system maintains an ECLIPSE MV/8000®stack for every task. When you load a program into an inner ring, the operating system allocates stack space for all its tasks, including those that are currently inactive; the operating system also initializes the stack pointers. All stacks are the same size; that is, they are the size that you specified when you linked the new program.

Notes

- See the descriptions of ?RNGPR and ?RNGST in this chapter.
- Refer to the *ECLIPSE® MV/Family (32-Bit) Systems Principles of Operation* manual for information on gates and gate arrays.

?RNAME — Determines whether a pathname contains a reference to a remote host.

?RNAME
error return
normal return

Input

AC0	Byte pointer to pathname
AC1	Reserved (Set to 0.)
AC2	Reserved (Set to 0.)

Output

AC0	One of the following: • 0 if host reference is local • –1 if host reference is remote and host ID is 0 • Host ID in Bits 17 through 31
AC1	Undefined
AC2	Undefined

Error Codes in AC0

File system error codes

Why Use It?

?RNAME allows you to detect remote references in pathnames.

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

Given a pathname, ?RNAME returns the host ID of the host on which the file should exist. Note that this does not mean that the file does exist.

If ?RNAME returns a host ID of –1 in Bits 17 through 31 of AC0, the pathname referenced a remote host with a host ID of 0.

?RNGPR
error return
normal return

Input

AC0 One of the following:

- PID
- Byte pointer to the process name
- For information about self, –1

AC1 One of the following:

- –1 if AC0 contains a byte pointer to the process name
- Any number other than –1 if there is a PID or a –1 in AC0

AC2 Word pointer to the ?RNGPR packet

Output

Length of the string returned (in offset ?RNGLB.)

Error Codes in AC0

ERIRB	Insufficient room in buffer
ERPRH	Process not in hierarchy
ERRNI	Ring number invalid
ERRNL	Ring not loaded
ERVBP	Illegal byte pointer
ERVWP	Illegal word pointer

Why Use It?

You can use ?RNGPR to find out what program the operating system has loaded into a specific ring or whether it has loaded a program at all.

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

?RNGPR allows a process to verify which program has been loaded into a specific ring. If the ring has not already been loaded, the operating system returns error code ERRNL in AC0.

You can request this information for the calling process, for a specified PID, or by specifying a process name. The buffer to which the caller's packet points never exceeds ?MXPL words.

To use ?RNGPR, you must supply a packet that contains a byte pointer to the buffer and the number of the ring you want to check. Figure 2–208 shows the structure and Table 2–186 lists the contents of the ?RNGPR packet.

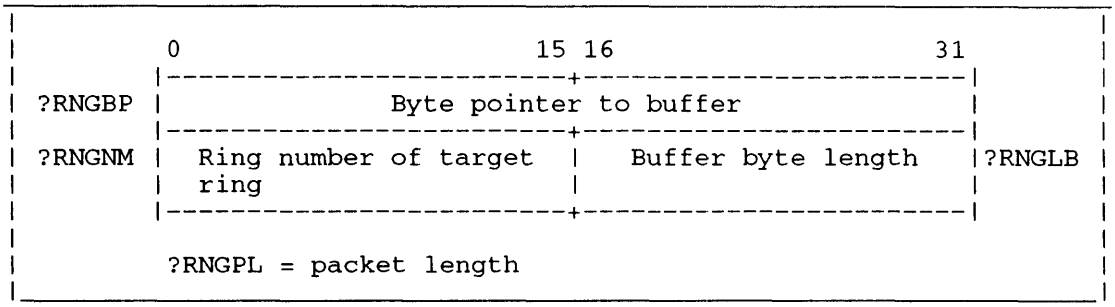


Figure 2–208. Structure of ?RNGPR Packet

Table 2–186. Contents of ?RNGPR Packet*

Offset	Contents
?RNGBP (doubleword)	Byte pointer to buffer.
?RNGNM	Number of ring to check.
?RNGLB	Length of buffer (in bytes).

* There is no default unless otherwise specified.

Notes

- See the descriptions of ?RINGLD and ?RINGST in this chapter.

?RNGST — Stops ?RINGLD from loading lower rings.

?RNGST
error return
normal return

Input

AC0	Logical address of the lowest ring that allows loading
AC1	Reserved (Set to 0.)
AC2	Reserved (Set to 0.)

Output

AC0	Unchanged
AC1	Unchanged
AC2	Unchanged

Error Codes in AC0

ERPTY	Illegal process type
ERRNI	Ring number invalid
ERRAL	Ring already loaded

Why Use It?

?RNGST prevents a user's ?RINGLD from circumventing a ring's protection mechanisms.

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

?RNGST lets a process specify the lowest ring that ?RINGLD can load. ?RNGST allows servers to prevent themselves from being circumvented by a user process in Ring 7 that is loading a lower ring. If user rings below the specified address are already loaded, the operating system returns the error code ERRAL to AC0. Also, in that case, AC1 contains the ring number of the ring that failed.

A caller can specify that ?RINGLD cannot load programs into rings that are higher than the caller's own ring. However, in this case, the operating system ignores the fact that the caller's ring is already loaded.

The ring–maximization rules apply to ?RNGST. Therefore, an outer–ring ?RNGST cannot override an inner–ring ?RNGST.

?RNGST does not affect system rings.

Notes

- See the descriptions of ?RINGPR and ?RINGLD in this chapter.

?RPAGE
error return
normal return

Input

AC0 Contains the following:

- Bit 0 is a flag bit:

  If Bit 0 = 1, flush the contents of the page to disk before releasing it

  If Bit 0 = 0, do not flush the contents of the page to disk before releasing it

- Bits 1 through 31 contain any address in the target page (The OS converts this value to the correct logical page number.)

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?

?RPAGE removes a shared page from your logical address space, but does not necessarily remove it from memory. This eliminates the need for disk I/O, if another process requires the same page.

When you set the flag bit (Bit 0) in AC0, the ?RPAGE simulates a ?FLUSH; that is, the operating system retains the page in memory, but flushes its contents to disk if it is modified. This option guarantees that the shared page is updated immediately.

Who Can Use It?

There are no special process privileges needed to issue this call. The restrictions concerning file access are the same as those for system call ?SOPEN.

What It Does

?RPAGE releases a shared memory page from the caller's logical address space. If no other process is using the target page after the ?RPAGE, the operating system places it on the LRU chain for eventual reuse. (Note that the page remains in the user context; ?RPAGE invalidates the address for the user.)

If you set Bit 0 of AC0 and the page has been modified, the operating system flushes the contents of the page to disk. This variation is functionally the same as issuing ?FLUSH followed immediately by a standard ?RPAGE.

?RTODC — Reads the time–of–day conversion data.

AOS/VS

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

Input

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

Output

AC0	Unchanged
AC1	Unchanged
AC2	Address of the ?RTODC packet

Error Codes in AC0

ERICD	Illegal function code
ERPVS	Packet version not supported (bad value in ?RTODC_PKT.PKT_ID)
ERRVN	Reserved value not zero
ERVWP	Invalid word pointer passed as system call argument

Why Use It?

You can use the data that ?RTODC places in its parameter packet to interpret the numbers read from the system clock as local time or as Universal 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

This system call returns the current constants that the system uses to change the 64–bit number read from the system clock to either local time or to Universal Time. These constants, when added to a system clock time using 64–bit arithmetic, yield the number of system clock ticks since midnight, 1 January 1968 in the appropriate time zone. This system call also returns the current description for the local time zone (offset from Universal Time). Local time is typically the time at the site with the ECLIPSE MV/Family hardware on which the operating system is running.

The description of system call ?CLSTAT contains a summary of the 64–bit number that is in the system clock. Your system's "Principles of Operation" manual has additional information about clocks. However, we state here that the system clock ticks at 2.61244 gigahertz and that the first 48 bits are accurate. Bit 45. changes every 100 microseconds, which means it has a frequency of 10,000 hertz. (1 hertz = 1 Hz = 1 cycle per second.)

16–bit programs cannot issue ?RTODC.

Figure 2–209 shows the structure of the ?RTODC parameter packet, and Table 2–187 describes its contents.

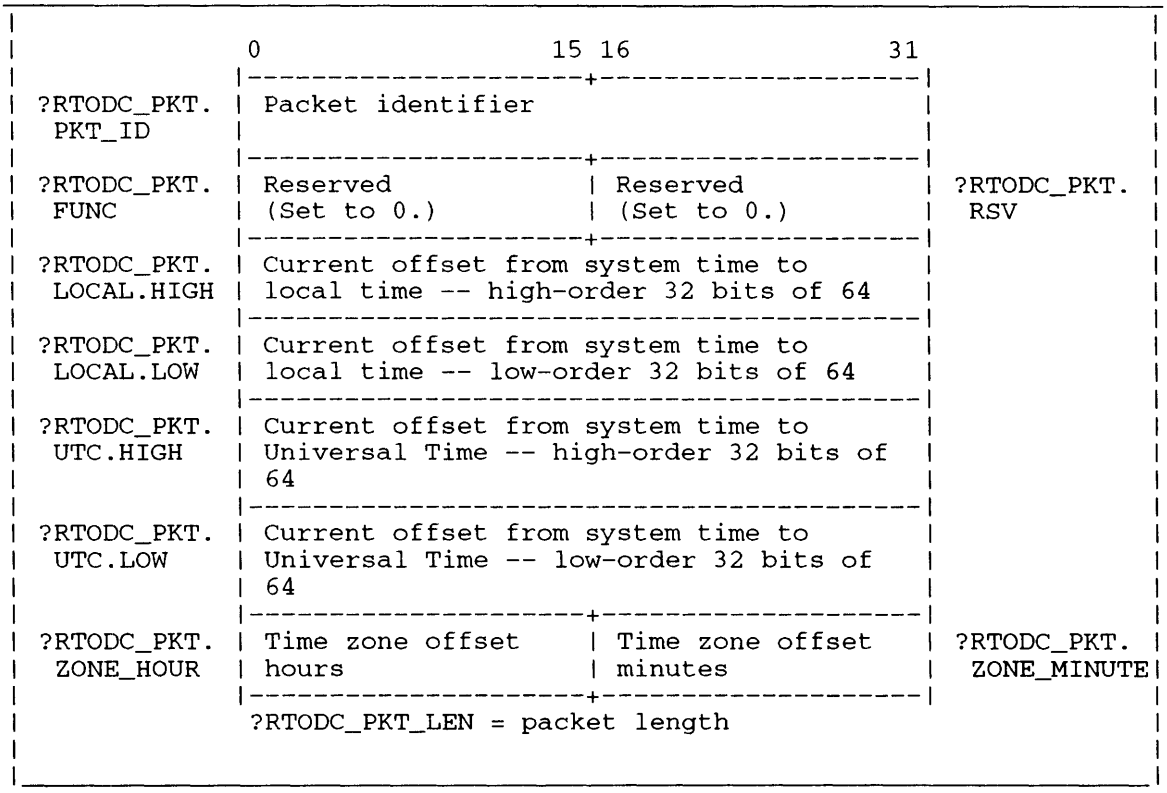


Figure 2–209. Structure of ?RTODC Packet

Table 2–187. Contents of ?RTODC Packet

Offset	Contents
?RTODC_PKT.PKT_ID (doubleword)	Packet identifier. Place ?RTODC_PKT_PKTID here.
?RTODC_PKT.FUNC	Reserved. (Set to 0.)
?RTODC_PKT.RSV	Reserved. (Set to 0.)
?RTODC_PKT. LOCAL.HIGH and ?RTODC_PKT. LOCAL.LOW	The operating system returns the respective high- and low-order parts of a 64-bit number which, when added to the current system clock, gives the number of system clock ticks since midnight, 1 January 1968 local time.
?RTODC_PKT. UTC.HIGH and ?RTODC_PKT. UTC.LOW	The operating system returns the respective high- and low-order parts of a 64-bit number which, when added to the current system clock, gives the number of system clock ticks since midnight, 1 January 1968 Universal time.
?RTODC_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). The numbers range between -12 and +13 inclusive.
?RTODC_PKT. ZONE_MINUTE	The operating system returns the current offset in minutes relative to ?RTODC_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, England it is 0830 (8:30 am) in India; consequently ?RTODC_PKT.ZONE_MINUTE contains 30 (decimal) and ?RTODC_PKT.ZONE_HOUR contains 5.

Notes

- See the descriptions of ?GTIME and ?NTIME in this chapter.

?RUNTM — Gets runtime statistics on a process.

?RUNTM *[packet address]*
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 obtain statistics for the calling process
AC1	One of the following: • –1 if AC0 contains a byte pointer • 0 if AC0 contains a PID Otherwise, the OS ignores AC1
AC2	Address of the ?RUNTM packet, unless you specify the address as an argument to ?RUNTM

Output

AC0	Unchanged
AC1	Unchanged
AC2	Unchanged

Error Codes in AC0

ERPRH	Attempt to access process not in hierarchy
ERVBP	Invalid byte pointer passed as a system call argument
ERVWP	Invalid word pointer passed as a system call argument

Why Use It?

Like ?PSTAT and ?WHIST, ?RUNTM returns information that you can use to judge a process's performance and its use of system resources.

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

?RUNTM returns runtime statistics for either the calling process or any other process the caller specifies. Before you issue ?RUNTM, load AC0 with the PID of the target process or with a byte pointer to its name. If you specify –1 in AC0, the operating system returns statistics for the calling process.

?RUNTM Continued

You must also reserve a packet of ?GRLTH words to receive the ?RUNTM statistics. You can cite the packet address as an argument to ?RUNTM or you can load its address into AC2 before you issue ?RUNTM. Figure 2–210 shows the packet structure.

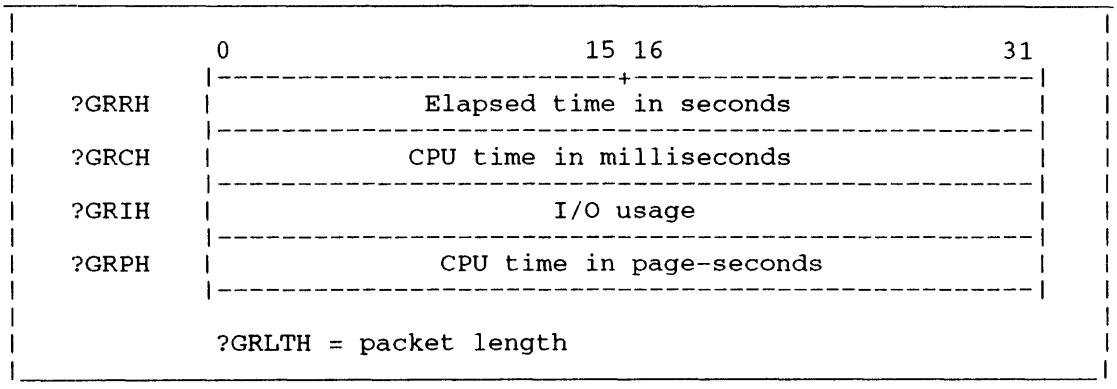


Figure 2–210. Structure of ?RUNTM Packet

As Figure 2–210 shows, the operating system returns the following information to the ?RUNTM packet: the elapsed time of the target process, its CPU time, its page usage over CPU time, and its I/O usage (number of blocks read or written).

Sample Packet

```
PKT:    .BLK    ?GRLTH          ;Allocate enough space for packet.
                                ;Packet length = ?GRLTH.
        .LOC    PKT+?GRRH       ;The OS will return elapsed time
        .DWORD  0               ;(in seconds).
        .LOC    PKT+?GRCH       ;The OS will return CPU time (in
                                ;milliseconds).
        .DWORD  0
        .LOC    PKT+?GRIH       ;The OS will return I/O usage
        .DWORD  0               ;information.
        .LOC    PKT+?GRPH       ;The OS will return page usage
        .DWORD  0               ;information.
        .LOC    PKT+?GRLTH      ;End of packet.
```

Notes

- See the descriptions of ?PSTAT and ?WHIST in this chapter.

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

Operating System Differences

Under AOS/RT32, ?SACL does nothing.

Input

AC0 One of the following:

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

AC1 One of the following:

- Byte pointer to the new ACL
- –1 to delete the current ACL

AC2 One of the following:

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

Output

AC0 Unchanged

AC1 Undefined

AC2 Undefined or address of ?SACL packet

Error Codes in AC0

ERACL Illegal ACL
ERVBP Invalid byte 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 file with ?ODTL value supplied in ?GOPEN packet

Why Use It?

?SACL allows you to alter or delete a file's current access control list (ACL). You can use ?SACL to change the ACL you set when you created the file.

The operating system reverts to the default access privileges when you issue ?DELETE, and then ?CREATE, against a file.

To find out what the ACL is for a particular file or directory before you issue ?SACL, issue ?GACL.

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 Owner access to the target file or Write access to the target file's parent directory. 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

Under AOS/RT32, ?SACL does nothing. Under AOS/VS, ?SACL replaces the target file's or directory's ACL with the ACL that you specify in AC1. If there is no new ACL, ?SACL deletes the existing ACL.

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

Before you issue ?SACL, define the new ACL in your address space and load AC1 with a byte pointer to its address. Structure the ACL specification as follows:

username<0><access privs>[username<0><access privs>...]<0>

For an example of 9 bytes:

OP<0><?FACO+?FACW+?FACA+?FACR+?FACE>CW<0><?FACR+?FACE><0>

where

access privs is one or more of the following access types:

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

The brackets ([ ]) in the format mean that you can repeat the

username<0><access privs>

entry. However, be sure to type the ACL specifications on one line. If you use a carriage return or New Line character, the operating system interprets that character as part of the ACL. Do not insert any spaces between the ACL mnemonics, and do terminate the ACL specification with a null byte.

To give a series of access privileges to all users, use the +<0> template before the ACL specification. For example, the specification

+<0><?FACR+?FACE><0>

gives all users Read and Execute access to the file. The symbol ?MXACL represents the maximum length for an ACL specification.

For another example, the CLI command

ACL FOO SAM,OAR LYNN,RE

sets the ACL of file FOO. You can also do this with the ?SACL system call. One step is to create a buffer containing the following 12. bytes in its leftmost bytes.

SAM<0><?FACO+?FACA+?FACR>LYNN<0><?FACR+?FACE><0>

Each of the symbols

<?FACO+?FACA+?FACR> and <?FACR+?FACE>

requires 1 byte.

Figure 2–211 shows the structure of the ?SACL packet.

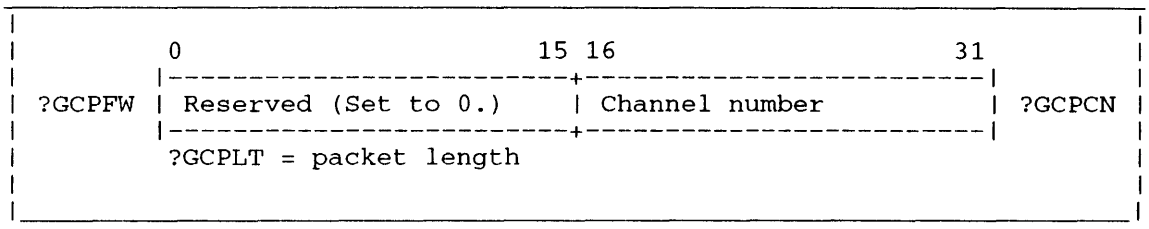


Figure 2–211. Structure of ?SACL Packet

Notes

- See the descriptions of ?CREATE, ?DELETE, and ?GACL in this chapter.

?SATR — Sets or removes the permanent attribute for a file or directory.

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

Input

AC0	One of the following: • Byte pointer to the pathname of the target file or directory • 0 if a packet address is supplied
AC1	Bit 31 is a flag bit: • Bit 31 = 1 to set the permanent attribute • Bit 31 = 0 to remove the PERMANENCE attribute
AC2	One of the following: • Reserved (Set to 0 if not supplying a packet.) • Address of the ?SATR packet, unless you specify the address as an argument to ?SATR

Output

AC0	Unchanged
AC1	Unchanged
AC2	Undefined or address of ?SATR packet

Error Codes in AC0

ERIFT	Illegal file type
ERVBP	Invalid byte 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)

Why Use It?

You can protect a file from being deleted by any process, regardless of the process's access privileges, by assigning that file the permanent attribute. If you assign the PERMANENCE attribute to a file, you should assign that attribute to its parent directory as well. Otherwise, a process can delete the file indirectly, simply by deleting the parent directory.

To find out if a file has the PERMANENCE attribute, issue ?FSTAT and examine the ?SSTS offset in the ?FSTAT return packet.

Who Can Use It?

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

What It Does

?SATR either assigns or removes the PERMANENCE attribute for a file or directory, depending on your input to AC1. You cannot delete any entries that have the PERMANENCE attribute. (The PERMANENCE attribute is the only file attribute that is currently defined.)

The target file 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 ?SATR packet. The use of the 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.

Figure 2–212 shows the structure of the ?SATR packet.

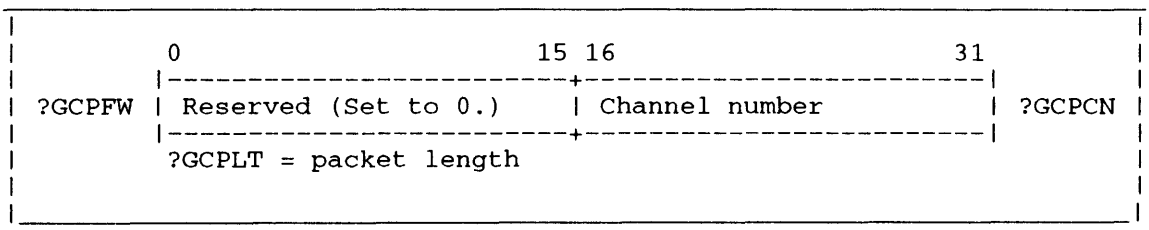


Figure 2–212. Structure of ?SATR Packet

Notes

- See the description of ?FSTAT in this chapter.

AOS/VS

?SBIAS
error return
normal return

Input

AC0	Contains the following: • Maximum bias value in Bits 16 through 23 • Minimum bias value in Bits 24 through 31
AC1	Reserved (Set to 0.)
AC2	Reserved (Set to 0.)

Output

AC0	Unchanged
AC1	Undefined
AC2	Undefined

Error Codes in AC0

ERBIF	Illegal bias factor
ERPRV	Caller not privileged for this action (The caller is not PID 2.)

Why Use It?

By setting maximum and minimum bias factors, you can influence the operating system's normal bias towards interactive processes (processes that interact with the operating system through a terminal). The system manager or operator usually sets the bias factors during the system–generation procedure.

To find out what current bias factor values have already been set for your system, issue ?GBIAS.

Who Can Use It?

Only the operator process (PID 2), or a process that has System Manager privilege, can issue ?SBIAS. There are no restrictions concerning file access.

What It Does

?SBIAS sets the maximum and minimum bias factors.

The minimum bias factor represents the fewest number of noninteractive processes the operating system will try to maintain in memory at one time. You must set a minimum bias factor; either with ?SBIAS or during the system–generation procedure. In general, the minimum bias factor should be low to avoid unfair bias toward noninteractive processes.

The maximum bias factor represents the maximum number of noninteractive processes the operating system will allow in memory before favoring blocked interactive processes. The maximum bias factor is optional.

If you do set the maximum bias factor, note that it is not absolute. Even after reaching the maximum bias value, the operating system can run another noninteractive processs if it can do so without pre–empting another process.

Notes

- See the description of ?GBIAS in this chapter.

?SCHR
error return
normal return

Input

AC0	One of the following: • Byte pointer to the device name • The number of the channel on which the device is open
AC1	Bits 0 and 1 are flag bits: • Bit 0 = 0 if AC0 contains a byte pointer • Bit 0 = 1 if AC0 contains the channel number • Bit 1 = 0 to change the device characteristics • Bit 1 = 1 to define default characteristics
AC2	Address of the three–word block to specify the device characteristics packet

Output

AC0	Unchanged
AC1	Unchanged
AC2	Unchanged

Error Codes in AC0

ERICN	Illegal channel
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?

The ?SCHR call allows the owner process to change the character device's current characteristics for the duration of the calling process. It enables the calling process to select certain characteristics dynamically to tailor the target device's I/O. The ?SECHR call is a functional superset of ?SCHR and can perform all ?SCHR's functions.

Who Can Use It?

You may use ?SCHR with or without special process privileges. Access to files differs with different process privileges, and these differences are described in the next section.

What It Does

Use the ?SCHR call to change the current or default characteristics for a character device. Clear bit 1 of AC1 to change the named device's current characteristics; set bit 1 to change the named device's default characteristics. As you indicate the length of the characteristics packet in AC1, you can use this call to set any number of characteristic words up to ?CLMAX. A process using ?SCHR can change specific characteristics words for a character device to which the caller has access. ?SCHR also allows the operator process (PID 2) or a process with the System Manager privilege ON to set new default characteristics for any character device, thus overriding the defaults chosen during the system generation dialog. ?SCHR allows

- the process that owns the target device to set the current characteristics of a character device.
- the operator process (PID 2) or a process with the System Manager privilege turned ON to set new default characteristics for any character device, and to override the defaults that were chosen during the system–generation procedure. The default values can be changed at any time. However, the new values will not take effect until the line is disabled (from EXEC) and enabled again.

If a process does not own a character device, it cannot change the current characteristics of a character device using ?SCHR. Before you issue ?SCHR, load AC0 with the channel number of the target device or with a byte pointer to the device name. Then, load AC2 with the address of the "new" device characteristics words, which you must supply elsewhere in your program. See Table 2–29 and Table 2–30, which are in the description of system call ?GCHR, for a list of the device characteristics words set by ?SCHR. For the complete list of characteristics device words set by ?SCHR, see Table 2–32 on page 2-194.

If AC0 contains a byte pointer, set Bit 0 of AC1 to 0; if AC0 contains the device's channel number, set Bit 0 of AC1 to 1.

You can also set what are known as "extended" characteristics for a character device. The flags in the words following the three that ?SCHR uses control XON/XOFF data flow over terminal lines and some of the operating characteristics of Intelligent Asynchronous Controllers (IACs). See the section "Additional Character Device Characteristics Words" in the explanation of system call ?GCHR.

To read the characteristics of a character device, use the ?GCHR and ?GECHR system calls.

It's possible to set the characters–per–line or lines–per–page value in characteristic word 2 at a value too large for the physical screen, but we don't recommend this.

Notes

- See ?SECHR in this chapter for setting extended characteristics.
- Refer to the chapter about VSGEN in the manual *Installing, Starting, and Stopping AOS/VS II* for information on setting operator process defaults.

?SCLOSE Closes a file previously opened for shared access.

?SCLOSE
error return
normal return

Input

AC0 Reserved (Set to 0.)

AC1 Contains the following:

- Bit 0 is a flag bit:

  Bit 0 = 1 to release the shared page(s) from the caller's logical address space

  Bit 0 = 0 to retain the shared page(s) from the caller's logical address space

- Bits 2 through 31 contain the file's channel number (returned by the OS as output to ?SOPEN)

AC2 Reserved (Set to 0.)

Output

AC0 Undefined

AC1 Unchanged

AC2 Undefined

Error Codes in AC0

ERCIU	Channel in use
ERFNO	Channel not open

Why Use It?

You must issue ?SCLOSE to close a shared file. However, you cannot use ?SCLOSE to close a file that you opened with ?OPEN or ?GOPEN.

After the operating system executes ?SCLOSE, it updates the file's status information and frees the channel so that it can be used again.

Who Can Use It?

There are no special process privileges needed to issue this call, and there are no restrictions concerning file access. However, you can issue ?SCLOSE against a file only if you have previously issued ?SOPEN against the file.

What It Does

?SCLOSE closes a file that ?SOPEN previously opened for shared access.

You can direct the operating system to release any pages of the target page that are still in the caller's logical address space by setting Bit 0 of AC1. If you do not set Bit 0 and pages of the file are still in the address space, the operating system takes the error return from ?SCLOSE and returns error code ERCIU in AC0.

Notes

- See the descriptions of ?SOPEN, ?OPEN, and ?GOPEN in this chapter.

?SDAY
error return
normal return

Input

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

Output

AC0	Unchanged
AC1	Unchanged
AC2	Unchanged

Error Codes in AC0

ERTIM	Illegal time argument (The day, month, or year value is outside the legal range.)
ERPRV	Caller not privileged for this action (A process other than the operator process attempted to set the day, month, or year.)

Why Use It?

?SDAY lets you (PID 2) adjust the operating system's internal calendar while your program is executing. Usually, the system manager or operator sets the calendar during the system–generation procedure.

Who Can Use It?

Only the operator process (PID 2), or a process with System Manager privilege, can issue ?SDAY. There are no restrictions concerning file access.

What It Does

?SDAY sets the system calendar to the day, month, and year you specify in AC0, AC1, and AC2, respectively. The operating system automatically increments the date when the time of day passes 23 hours, 59 minutes, 59 seconds.

?SDBL
error return
normal return

Input		Output	
AC0	Reserved (Set to 0.)	AC0	Undefined
AC1	Channel number assigned to the line	AC1	Unchanged
AC2	Reserved (Set to 0.)	AC2	Undefined

Error Codes in AC0

ERDSL	Device associated with the channel number is not a synchronous line

Why Use It?

You can use ?SDBL to disable a BSC (bisynchronous) line previously enabled with ?SEBL.

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

?SDBL disables the BSC line associated with the channel number that you specify in AC1. When you issue ?SDBL, the operating system breaks the association between the line and its channel, and the association between the line and the process that enabled it.

Notes

- See the description of ?SEBL in this chapter.

?SDLM
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 set the output delimiter table • ?SDTI to set the input delimiter table • ?SDTP to set the priority input delimiter table
AC2	Word address of the 16–word delimiter table

Output

AC0	Unchanged
AC1	Unchanged
AC2	Unchanged

Error Codes in AC0

ERICN	Illegal channel
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?

?SDLM allows you to override the default delimiters for data–sensitive records by defining a new delimiter table. In addition, ?SDLM overrides the delimiter table that you defined in the ?OPEN packet for the file. Thus, you can use ?SDLM to redefine the delimiter table you set up at ?OPEN 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

?SDLM sets a delimiter table for a file with the data–sensitive records or for a character device, while the file or device is open. ?SDLM is an alternative to the delimiter table feature in the ?OPEN packet.

Before you issue ?SDLM, set up a delimiter table of 16 consecutive words in your logical 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.)

If you load –1 into AC2, the operating system reverts to the default delimiters for data–sensitive files; that is, null = <000>, New Line = <012>, form feed = <014>, and carriage return = <015>. If you load 0 into AC2, the null character is the only valid delimiter.

If you issue ?SDLM for a character device, such as a terminal, you can define one of three types of delimiter tables; your input to AC1 defines the type. For an output delimiter table (a table that defines delimiters for output to the device), specify bit ?SDTO in AC1. For an input delimiter table (a table that defines delimiters for input to the device), specify bit ?SDTI in AC1. If you have opened the device with the priority read option, specify ?SDTP in AC1 to designate the table as a priority input delimiter table.

If you define a delimiter table for a file in the ?OPEN packet, and then issue ?SDLM, the new delimiter table overrides the delimiter table that you defined in the ?OPEN packet.

Should you want to save an earlier delimiter table before creating a new one with ?SDLM, the ?GDLM system call returns the delimiter table.

Notes

- See the description of ?GDLM and ?OPEN in this chapter.

?SDPOL — Defines a polling list or a poll–address/ select–address pair.

?SDPOL
error return
normal return

Input

AC0	One of the following values in Bits 16 through 23: • Length of each poll address entry (including device address characters), if the caller is a control station • Length of the poll address, if the caller is a tributary (the select address is the same length) One of the following values in Bits 24 through 31: • Number of relative terminals (tributaries) in the polling list, if the caller is a control station • 1 if the caller is a tributary
AC1	Channel number assigned to the line
AC2	Byte pointer to the polling list

Output

AC0	Unchanged
AC1	Unchanged
AC2	Unchanged

Error Codes in AC0

ERIMM	Not enough memory for poll/select list
ERLNA	I/O request for disabled line
ERLNM	Line not multipoint
ERNSL	Attempt to enable nonsynchronous line

Why Use It?

A control station must define a polling list with ?SDPOL before it can perform polling or selecting. Similarly, a tributary station must define its poll and select addresses with ?SDPOL before its control station can poll or select 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

?SDPOL performs the following functions:

- If the caller is a control station on a multipoint line, ?SDPOL defines a polling list.
- If the caller is a tributary on a multipoint line, ?SDPOL defines the caller's poll address and select address.

Note that only stations on a multipoint bisynchronous communications (BSC) line can issue ?SDPOL.

The operating system uses the addresses that ?SDPOL defines during receiving initial (?SRCV) general polling sequences. If your computer is a control station, you must issue ?SDPOL before you can perform polling or selecting. If your computer is a tributary, you must issue ?SDPOL to define your poll and select addresses.

Defining a Polling List

If you use ?SDPOL from a control station, construct the polling list in contiguous locations in your logical address space before you issue ?SDPOL. The entries in the list can be of any byte length, provided each entry is the same length. Figure 2–213 shows a sample polling list, where each entry is 4 bytes long.

Before you issue ?SDPOL, load AC0 with the byte length of the entries (Bits 16 through 23) and the number of entries (Bits 24 through 31). The operating system uses these two values to determine the length of the polling list and to assign relative terminal numbers to the tributaries.

Defining a Poll Address and Select Address for a Tributary

The procedure for defining a single poll address and select address entry for a tributary is similar to the procedure for defining a polling list. First, define the poll and select addresses in contiguous locations in your logical address space. Then, before you issue ?SDPOL, load Bits 16 through 23 of AC0 with the byte length of the poll or select address. (Each address must be the same byte length.) Load Bits 24 through 31 of AC0 with 1. Figure 2–214 shows a sample poll address and select address entry, where each address is 2 bytes.

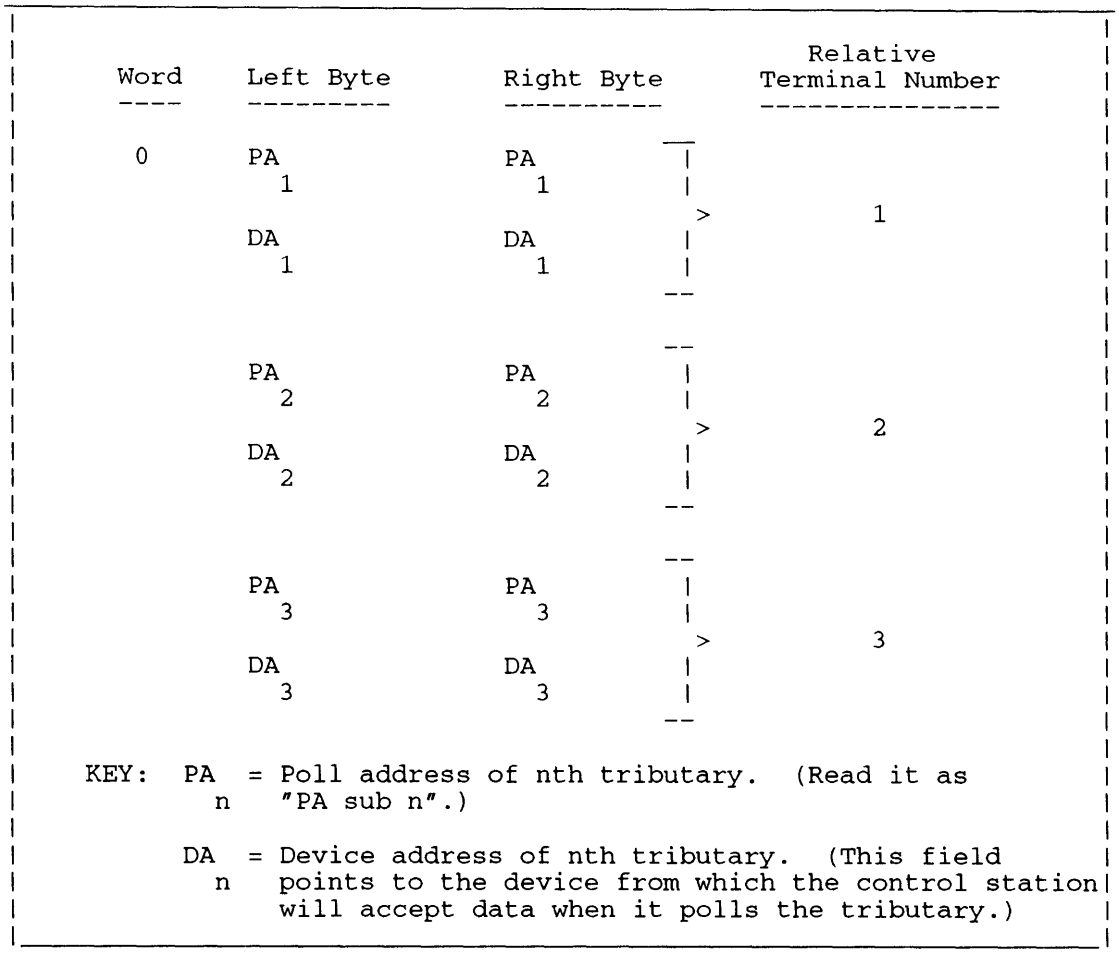


Figure 2–213. Polling List Defined by a Control Station

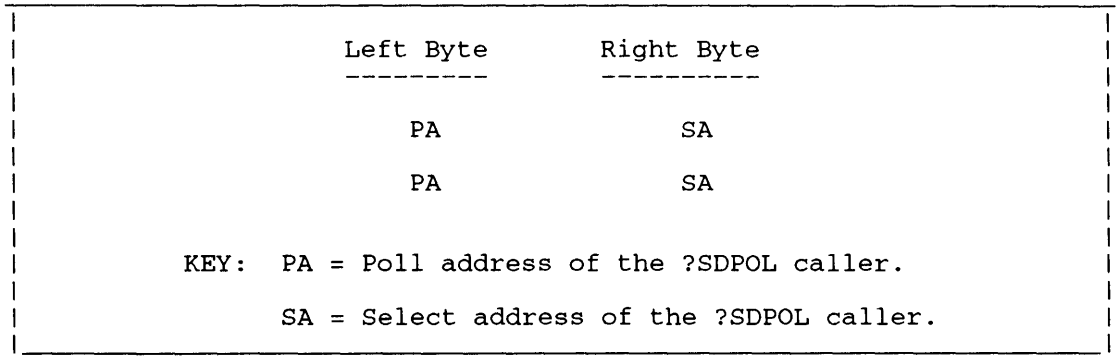


Figure 2–214. Poll and Select Addresses Defined by a Tributary

Notes

- See the description of ?SRCV in this chapter for more information on polling.

?SDRT
error return
normal return

?SERT
error return
normal return

Input

AC0	One of the following values in Bits 16 through 23: • Length of each poll address entry (including device address characters), if the caller is a control station • Length of the poll address, if the caller is a tributary (the select address is the same length) One of the following values in Bits 24 through 31: • Number of relative terminals (tributaries) in the polling list, if the caller is a control station • 1 if the caller is a tributary
AC1	Channel number assigned to the line
AC2	Reserved (Set to 0.)

Output

AC0	Unchanged
AC1	Unchanged
AC2	Unchanged

Error Codes in AC0

ERIRT	Illegal relative terminal number
ERLNA	I/O request for disabled line (?SERT)
ERLNM	Line not multipoint
ERNPL	Polling list not defined
ERNSL	Attempt to enable nonsynchronous line (?SERT)

Why Use It?

When a control station defines a polling list with ?SDPOL, the operating system automatically enables all the relative terminals on the list. In effect, ?SDRT "removes" a relative terminal from the polling list by directing the operating system to ignore that terminal during the next polling sequence. In general, you disable a tributary if it repeatedly fails to respond to a poll or to a select sequence.

?SDRT/?SERT Continued

?SERT allows the control station to "undo" a previous ?SDRT; that is, to re–enable a previously disabled relative terminal.

Disabling a relative terminal does not affect the corresponding tributary station. Instead, the control station ignores the tributary station until you subsequently re–enable the relative terminal or define a new polling 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

?SDRT and ?SERT respectively disable and re–enable a relative terminal for subsequent polling or selecting. Only a control station on a multipoint line can issue ?SDRT and ?SERT.

Notes

- See the description of ?SCRV in this chapter for more information on polling.

?SEBL
error return
normal return

Input		Output	
AC0	Byte pointer to the line's device name	AC0	Unchanged
AC1	Reserved (Set to 0.)	AC1	Undefined
AC2	Address of the ?SEBL packet	AC2	Unchanged

Error Codes in AC0

ERDCU	DCU inoperative (can't be initialized)
EREPE	Line already enabled on ?SEBL call
ERFCT	Failure to connect
ERINE	Initialization parameter error (You passed an invalid packet address.)
ERNSL	Attempt to enable nonsynchronous line

Why Use It?

You must use ?SEBL to enable a BSC line before you can use it.

If your computer is a control station on a multipoint line, ?SEBL enables the common line for you and your tributaries to use.

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

?SEBL enables a bisynchronous (BSC) line and associates it with the calling process. ?SEBL also directs the operating system to acquire resources for the line and to perform hardware–related initialization.

Before you issue ?SEBL, set up the ?SEBL packet in your logical address space, load AC2 with the packet address, and load AC0 with a device name for the line.

The device name for a BSC line is

@SLNx

where

@SLN	is the BSC designator.
x	is the line number.

Figure 2–215 shows the structure of the ?SEBL packet, and Table 2–188 describes the contents of each offset.

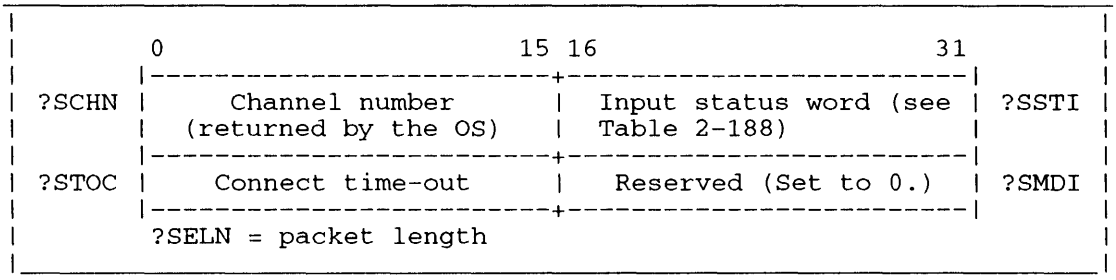


Figure 2–215. Structure of ?SEBL Packet

The Input Status Word

Offset ?SSTI contains specifications for the configuration, status, character set, parity, and protocol of the line you enable.

If you enable a point–to–point line (you selected ?SDPP as the line type), set the line mode to ?SDPR if your computer is the primary station. If your computer is the secondary station, set the line mode to ?SDSC. However, if you enable a multipoint line, set the line mode to ?SDPR if your computer is the control station and to ?SDSC if it is a tributary.

Table 2–188. Contents of ?SEBL Packet*

Offset	Contents
?SCHN	Channel number (returned by the OS).
?SSTI	Input status word.
	Line type. ?SDPP--Point-to-point. ?SDMD--Multipoint. DEFAULT = ?SDPP.
	Line mode. ?SDSC--secondary (point-to-point), tributary (multipoint). ?SDPR--primary (point-to-point), control station (multipoint). DEFAULT = ?SDSC.
	Line code. ?SEBC--EBCDIC. ?SASC--ASCII. DEFAULT = ?SEBC.
	Block Check. ?SCRC--CRC16. ?SCIT--CCITT16. ?SLRC--LRC. DEFAULT = ?SCRC.
	Parity. ?SNPR--No parity. ?SOPR--Odd parity. ?SEPR--Even parity. DEFAULT = ?SNPR.

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

Table 2–188. Contents of ?SEBL Packet*

Offset	Contents
?SSTI (cont.)	Protocol. ?SBSC--Normal BSC protocol. ?SHSP--HASP multileaving. DEFAULT = ?SBSC.
	Send station identification on a point-to-point line. ?ENID--Enable the sending or receiving of station IDs. You can have only one successful ID bid/response on each line you enable. DEFAULT = No station identification.
?STOC	Connect time-out value, in seconds. -1 30 seconds. 0 No time-out. 16383 Maximum time-out. DEFAULT = -1 (30 seconds).
?SMDI	Reserved. (Set to 0.)

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

The line code tells the operating system which character set, EBCDIC or ASCII, to use for the data–link control characters. The default character set is EBCDIC. Note that the operating system does not translate text or header information into the character set you specify here. You must translate the outgoing data, if necessary. However, the operating system returns incoming data to you in the character set in which it was received.

The block check field lets you select the block check scheme you want to use for the transmission. Block checking is the method that the operating system uses to validate the entire text block, according to the BSC protocol. (See Table 2–189 for a description of the block check data–link character.) The sending and receiving stations on a BSC line must use the same block check method. Otherwise, the operating system cannot verify the contents of the transmission blocks.

If your line code is EBCDIC (you selected ?SEBC or did not specify the line code), choose either ?SCRC or ?SCIT as the block check type. Both masks correspond to cyclical redundancy checking. (See Table 2–189.) ?SCRC corresponds to the CRC16 method and ?SCIT corresponds to the CCITT16 method.

Station Identification

You can send an identification number (ID) that precedes your line bid on a point–to–point line. The receiving station responds with its own ID or with an EOT. Then you act according to the receiving station's response.

System calls ?SEBL, ?SSND, and ?SRCV implement station identification via their respective bits ?ENID, ?SNID, and ?SRID in their input status words. See these system call descriptions and Figure 2–216.

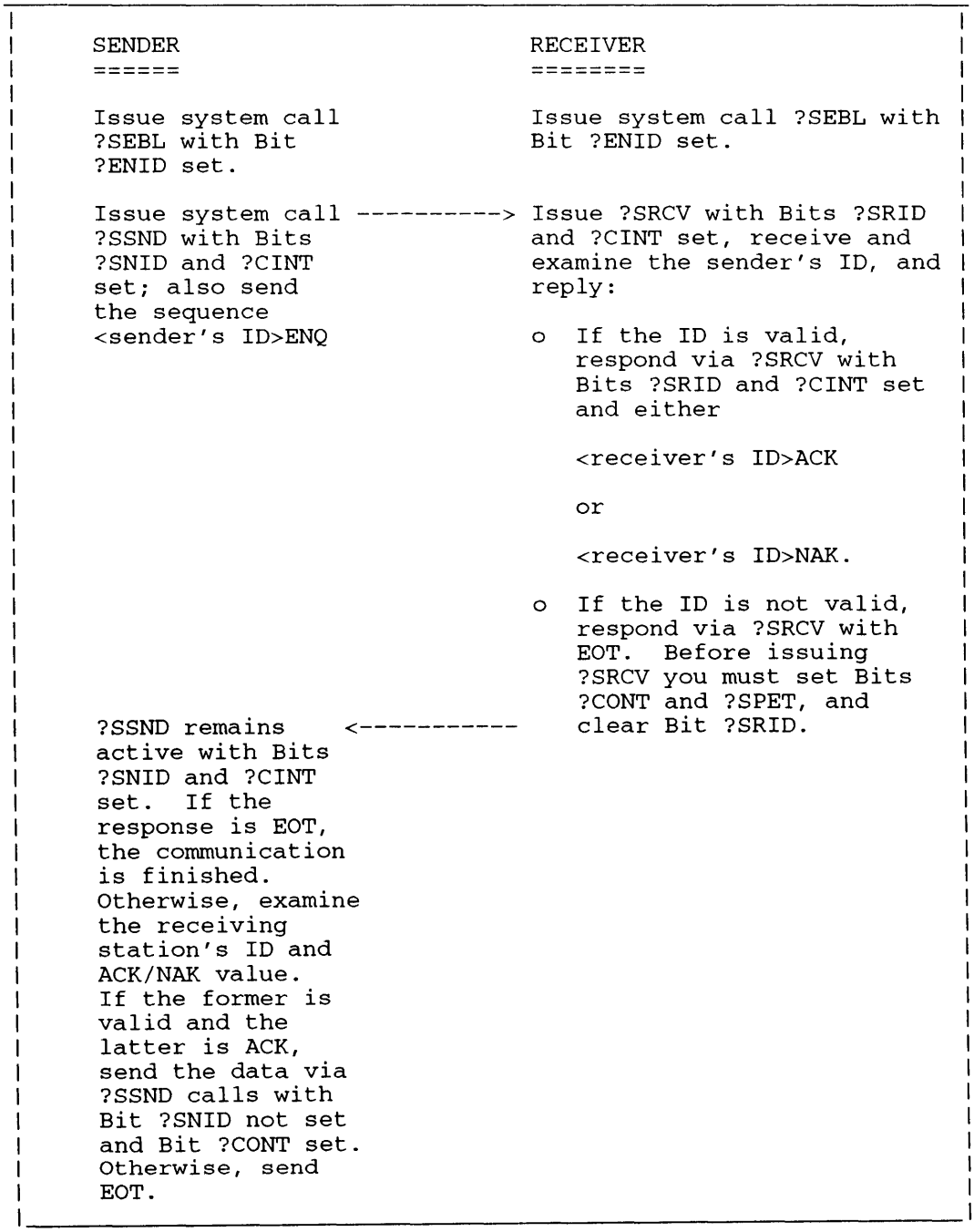


Figure 2–216. Station Identification System Call Sequence

Table 2–189. BSC Protocol Data–Link Control Characters (DLCC)

Character	Description
ACK0 ACK1	Affirmative Acknowledgment. Positive replies, sent in alternating sequence, to indicate that the receiver has accepted the previous block without error, and is ready to accept the next block of the transmission. ACK0 is also a positive response to a line bid (ENQ) for a point-to-point line and to a selection sequence on a multipoint line.
BCC	Block Check Character. A value generated by the transmitting station and sent with each data block to validate the block's contents. The receiving station generates its own BCC. If the two values match, the block is accepted as error-free. A BCC follows every ITB, ETB, and ETX character. If you transmit in the ASCII code set, the BCC is a longitudinal redundancy check (LRC). For the EBCDIC code set, the BCC is a cyclical redundancy check (CRC).
DLE	Data-Link Escape. The first character in a two-character sequence used to signal the beginning or end of Transparent Text mode. The sequence DLE STX signals the beginning of Transparent Text mode. The sequence DLE ETB or DLE ETX signals the end of Transparent Text mode.
DLE EOT	Data-Link Escape, End-of-Transmission. Signals a line disconnect for a switched line. The sending or receiving station usually transmits this sequence when all message exchanges are finished.

(continued)

Table 2–189. BSC Protocol Data–Link Control Characters (DLCC)

Character	Description
ENQ	Enquiry. Sent by a station on a point-to-point line to bid for the line (for transmission of data). Sent by the control station on a multipoint line to signal the end of a polling or selecting sequence. A transmitting station can also send ENQ to ask the receiver to repeat a response if the original response was garbled or not received when expected.
EOT	End-of-Transmission. Signals the end of a message transmission (consisting of one or more separately transmitted data blocks), and resets the receiving station. On a multipoint line, a polled station sends an EOT to indicate that it has nothing to send back to the control station. EOT can also serve as an abort signal to indicate a system or transmission malfunction.
ETB	End-of-Transmission Block.
ETX	End-of-Text. Signal the end of a data block that began with an SOH or an STX. Both the ETB and the ETX characters reverse the direction of the transmission. When a station receives an ETB or an ETX, it replies with a control character that indicates its status (that is, ACK0, ACK1, NAK, WACK, or RVI). An ETB terminates every text block except the last. An ETX implies an end-of-file condition; thus, it terminates the last block of text in a message.

(continued)

Table 2–189. BSC Protocol Data–Link Control Characters (DLCC)

Character	Description
ITB	End-of-Intermediate-Transmission Block. Separates records within a block and/or delimits field boundaries for error checking. ITB does not reverse the direction of the transmission.
NAK	Negative Acknowledgment. Sent by the receiving station to indicate that it is not ready to receive, or to request that an erroneous block be transmitted again.
RVI	Reverse Interrupt. A positive response used instead of ACK0 or ACK1; signals that the receiver must interrupt the transmission to send the transmitting station a high-priority message. The transmitting station treats an RVI as a positive acknowledgment and, in response, transmits all the data that prevents it from becoming a receiving station. The transmitting station can perform more than one block transmission to empty all its buffers. On a multipoint line, a control station can send an RVI after it receives data from a tributary station, to indicate that it wants to communicate with a different tributary station.
SOH	Start-of-Header. Signals the start of header information (ancillary information within a block).
STX	Start of Text. Signals the beginning of the text (and terminates the header information).

(continued)

Table 2–189. BSC Protocol Data–Link Control Characters (DLCC)

Character	Description
SYN	Synchronization Character. Establishes and maintains character synchronization; also serves as filler in the absence of data or control characters. Each transmission must begin with at least two contiguous SYN characters.
TTD	Temporary Text Delay. A two-character sequence that consists of STX ENQ, which the transmitting station sends to retain the line without immediately sending data. The receiving station responds with a NAK. The TTD/NAK sequence can repeat, if the transmitter needs additional delays.
WACK	Wait-Before-Transmit Positive Acknowledgment. A positive acknowledgment that the receiver sends; signals that the receiver is temporarily unable to receive. (A receiver can send WACK as a response to a line bid on a point-to-point line or a selection sequence on a multipoint line, or as a response to data.) A receiving station can send more than one WACK until it is ready to receive. The transmitting station can respond with ENQ, EOT, or DLE EOT.

(concluded)

If your line code is ASCII (you selected ?SASC as the line code), choose ?SLRC as the block check type. This block check type is illegal for EBCDIC transmissions.

The parity specification applies to transmission in ASCII code only. This parameter tells the operating system whether to check the parity of each character and whether to use odd parity (?SOPR) or even parity (?SEPR). If you set parity to ?SNPR, there is no parity checking.

Connect Time–out

The connect time–out specification, offset ?STOC, defines how many seconds the operating system will wait for the line to connect before it takes the ?SEBL error return. When you enable a BSC line, the operating system first examines the state of the data set flag, DSR, which the hardware raises as soon as the line is connected. If the DSR flag is not up, the operating system suspends the ?SEBL calling task until the DSR flag is raised. If the flag is not raised within the time–out interval, the operating system takes the ?SEBL error return and passes error code ERFCT to AC0.

?SEBL Continued

Sample Packet

```
PKT:    .BLK    ?SELN           ;Allocate enough space for the
                                ;packet.  Packet length = ?SELN.
        .LOC    PKT+?SCHN       ;Channel number.
        .WORD   0               ;The OS returns this value.
        .LOC    PKT+?SSTI       ;Input status word.
        .WORD   ?SDPP!?SDPR!?SASC!?SLRC!?SEPR!?SSBSC   ;Point-to-point
                                ;line type and mode, ASCII
                                ;format, even parity,
                                ;and normal BSC protocol.
        .LOC    PKT+?STOC       ;Connect time-out.
        .WORD   -1              ;30-second connect time-out (default
                                ;is -1).
        .LOC    PKT+?SMDI       ;Reserved.
        .WORD   0               ;You must set this value to 0.
        .LOC    PKT+?SELN       ;End of packet.
```

Notes

- See the description of ?SDBL in this chapter to disable a BSC line.

?SECHR
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 set the current characteristics
Bit 1 = 1	To set 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 specify the new 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.

Why Use It?

The ?SECHR call allows the owner process to change the character device's current characteristics for the duration of the calling process. It enables the calling process to select certain characteristics dynamically to tailor the target device's I/O. The ?SECHR call is a functional superset of the ?SCHR call.

?SECHR Continued

Who Can Use It?

You may use ?SECHR with or without special process privileges. Access to files differs with different process privileges, and these access differences are described in the next section.

What It Does

Use the ?SECHR call to change the current or default characteristics for a character device. Clear bit 1 of AC1 to change the named device's current characteristics; set bit 1 to change the named device's default characteristics. As you indicate the length of the characteristics packet in AC1, you can use this call to set any number of characteristic words up to ?CLMAX. A process using ?SECHR can change specific characteristics words for a character device(s) to which the caller has access. ?SECHR also allows the operator process (PID 2) or a process with the System Manager privilege ON to set new default characteristics for any character device, thus overriding the defaults chosen during the system generation dialog. ?SECHR allows

- the process that owns the target device to set the current characteristics of a character device.
- the operator process (PID 2) or a process with the System Manager privilege turned ON to set new default characteristics for any character device, and to override the defaults that were chosen during the system–generation procedure. The default values can be changed at any time. However, the new values will not take effect until the line is disabled (from EXEC) and enabled again.

No process can define new current characteristics for modem control (?CMOD) or monitor ring indicator (?CMRI) characteristics. The system manager can only change these characteristics by changing the default values. If a process does not own a character device, it cannot change the current characteristics of a character device using ?SECHR.

No process may use ?SECHR to change the split baud enable/disable characteristic: (?CSBEN/?CSBDS) or the two modem–related characteristics: modem control (?CMOD) and monitor ring indicator (?CMRI). These characteristics may only be set in the system generation dialog. Also there are characteristics fields which have no meaning on certain hardware configurations, and an attempt to set these fields will have no effect.

Before issuing ?SECHR, load AC0 with the channel number of the target device or with a byte pointer to the device name. Load AC1 with the number of characteristics words to be changed. Load AC2 with the address of the "new" device characteristics words, supplied elsewhere in your program. Table 2–32, in the description of system call ?GECHR, contains a list of character device characteristics for words zero through thirteen. File PARU.32.SR contains the symbol list of these words. Use a text editor (in Read mode) to open file PARU.32.SR; then find symbol "?CNNL" and move slowly forward from there until you encounter the offsets you need.

If AC0 contains a byte pointer, set Bit 0 of AC1 to zero; if AC0 contains the device's channel number, set Bit 0 of AC1 to 1.

You use the ?TRPE characteristic to indicate that pointer events (e.g., mouse movements) should delimit/terminate subsequent ?READs. If you are interested in retrieving pointer events, then set this characteristic.

It's possible to set the characters–per–line or lines–per–page value in characteristic word 2 at a value too large for the physical screen, but we don't recommend this.

?SEND
error return
normal return

Input

AC0	One of the following: • PID of the process associated with the terminal • Byte pointer to the name of the process associated with the terminal • Byte pointer to the name of the terminal
AC1	Byte pointer to the message
AC2	Contains the following: • Bits 22 and 23 are flag bits with the following binary values: 00 if AC0 contains a PID 01 if AC0 contains a byte pointer to the process name 10 if AC0 contains a byte pointer to the terminal name 11 (undefined) • Bits 24 through 31 contain the byte length (to a maximum of 255 bytes)

Output

AC0	Unchanged
AC1	Unchanged
AC2	Unchanged
	of the message

Error Codes in AC0

ERIDT	Illegal device name type (The target device is not a terminal.)
ERMEM	Insufficient memory
ERMRD	Message receive disabled
ERPRE	Invalid system call parameter
ERVBP	Invalid byte pointer passed as a system call argument

Why Use It?

?SEND allows a process to send a message to a console that a process (yours or another user's) owns. Two common uses if ?SEND are:

- To allow a process without a console file (perhaps a service–providing process) to send messages to the operator/service manager and to a client of the service.
- To allow an interactive process to accept a command.

?SEND Continued

Who Can Use It?

Except for overriding the ?CNRM characteristic, no special process privileges are needed to issue this call. The PID 2 process or a process with the System Manager privilege ON can override the ?CNRM characteristic. There are no restrictions concerning file access.

What It Does

?SEND transmits a message from a process to a console which, in turn, must be a process's console file. The console file cannot have the ?CNRM characteristic (message receive disabled) and it cannot be waiting for a Ctrl–Q because of page mode or a Ctrl–S. (The PID 2 process or a process with the System Manager privilege ON overrides the ?CNRM characteristic.)

Before you issue ?SEND, load AC1 with a byte pointer to the message text, which may be up to 255 bytes. The operating system sends the message in the following format:

<New Line>From PID xxxxx: <message><New Line>

where

PID xxxxx is the PID of the process issuing the ?SEND.

<message> is the message string.

You must specify the receiving terminal in AC0 before you issue ?SEND. To do this, supply one of the following in AC0:

- The PID of the process associated with the receiving terminal.
- The process name of the process associated with the receiving terminal (username:process_name).
- The simple pathname of the receiving device.

How you specify the receiving terminal determines your input to AC2.

?SERMSG — Returns text for associated error code (16–bit processes only).

?SERMSG
error return
normal return

Input

AC0	Reserved (Set to 0.)
AC1	Reserved (Set to 0.)
AC2	Address of the ?SERMSG packet

Output

AC0	Length of the message in bytes
AC1	Channel number
AC2	Unchanged

Error Codes in AC0

ERICM	32–bit program incorrectly attempted to call ?SERMSG
EVTXT	Caller's input message buffer is too small
ERVBP	Invalid byte pointer passed as a system call argument
ERVWP	Invalid word pointer passed as a system call argument

Why Use It?

?SERMSG allows a 16–bit program to refer to 32–bit error messages.

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

?SERMSG permits 16–bit programs to return the text string associated with a 32–bit error code. Sixteen–bit programs can use ?ERMSG to return the text string associated with a 16–bit error code.

To use ?SERMSG, you must supply a packet. Figure 2–217 shows the structure and Table 2–190 lists the contents of the ?SERMSG packet.

Upon return from ?SERMSG, AC0 and AC1 contain the same return information as from ?ERMSG.

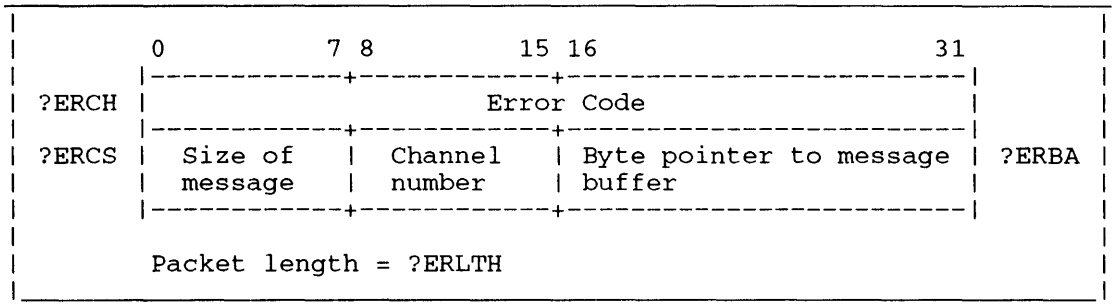


Figure 2–217. Structure of the ?SERMSG Packet

Table 2–190. Contents of the ?SERMSG Packet*

Offset	Contents
?ERCH (doubleword)	Error code.
?ERCS	Left byte: Size of message buffer. Right byte: Channel number. (A 377 specifies the system default.)
?ERBA	Byte pointer to message buffer.

* There is no default unless otherwise specified.

Notes

- See the description of ?ERMSG in this chapter.

?SERT — Re–enables a relative terminal.

?SERT
error return
normal return

See the description of ?SDRT in this chapter for more information on ?SERT.

?SERVE — Becomes a server.

?SERVE
error return
normal return

Input

AC0	None
AC1	None
AC2	None

Output

AC0	Unchanged
AC1	Unchanged
AC2	Unchanged

Error Codes in AC0

No error codes are currently defined.

Why Use It?

?SERVE and ?CON are the fundamental connection–management system calls. Potential servers must issue ?SERVE to declare themselves as servers. Similarly, potential customers must issue ?CON to establish connections with existing server processes.

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

?SERVE designates the calling process as a server, which can perform functions on behalf of customer processes. (Potential customers connect with the server by issuing ?CON system calls.)

Notes

- See the description of the ?CON system call in this chapter.

?SGES
error return
normal return

Input

AC0	For a cumulative error tally, –1; otherwise, ignored
AC1	Channel number assigned to the line
AC2	Address of the ?SGES packet

Output

AC0	Undefined
AC1	Unchanged
AC2	Unchanged

Error Codes in AC0

ERLNA	I/O request for disabled line
ERNSL	Attempt to enable nonsynchronous line

Why Use It?

?SGES helps you to pinpoint problem areas on each enabled BSC line.

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

?SGES returns the error statistics that the operating system records for each enabled bisynchronous (BSC) line. These statistics include the number of block check errors, the number of time–outs, and the total number of negative acknowledgment (NAK) characters received in response to send operations. If you set AC0 to –1 on input, the operating system returns a cumulative error tally. A cumulative error tally is a record of all errors that have occurred since the line was enabled or since the last ?SGES in which you did not select the cumulative error option.

Before you issue ?SGES, set up a packet of ?SGLN words, and load the packet address into AC2. Figure 2–218 shows the structure of the ?SGES packet. Table 2–191 describes each offset.

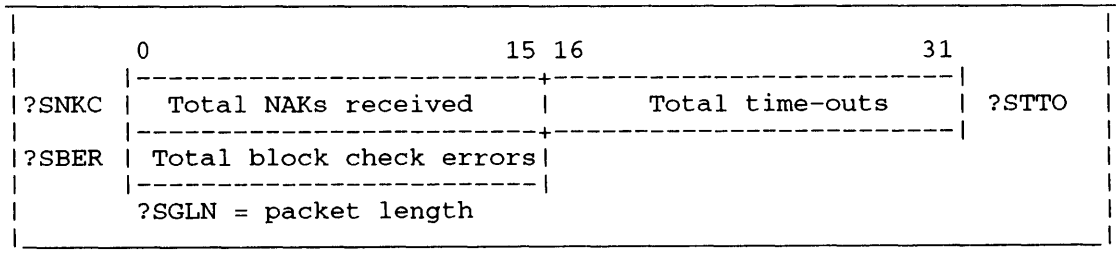


Figure 2–218. Structure of ?SGES Packet

Table 2–191. Contents of ?SGES Packet

Offset	Contents
?SNKC	Total number of NAKs received in response to a send operation. A large number of NAKs can indicate that the data was not received or, if it was received, that the OS found a block check error. (See the description of the ?SSND system call in this chapter.)
?SSTO	Total number of time-outs during receive operations. If there are no time-outs, the OS returns 0 to this offset. (See the description of the ?SRCV system call in this chapter.)
?SBER	Total number of block check errors during receive operations. These are not errors that lead to error returns from ?SRCV; rather, they are errors in the data. When an ?SRCV error occurs, the OS enters an error-recovery procedure in which it tries again to receive the data, repeatedly, if necessary. The OS may receive the data correctly on a retry.

* There is no default unless otherwise specified.

Sample Packet

```
PKT:    .BLK    ?SGLN           ;Allocate enough space for the
                                ;packet.  Packet length = ?SGLN.
        .LOC    PKT+?SNKC       ;Total number of NAKs received in
        .WORD   0               ;response to a send.  The OS
                                ;returns this value.  (A large number
                                ;of NAKs means that either the send
                                ;was not received or there is a block
                                ;check error.)
        .LOC    PKT+?STTO       ;Total number of time-outs during
        .WORD   0               ;receive.  The OS returns this value.
        .LOC    PKT+?SBER       ;Total number of block check errors
        .WORD   0               ;that occurred during the receive.
                                ;(These are errors in the data.  See
                                ;Table 2-191.)  The OS returns this
                                ;value.
        .LOC    PKT+?SGLN       ;End of packet.
```

Notes

- See the description of ?SCRV in this chapter for more information on receive operations.

?SIGNL
error return
normal return

Input

AC0	Reserved (Set to 0.)
AC1	PID of the target process (The caller can specify its own PID; however, the OS will validate whatever PID is specified.)
AC2	Unique TID of the target task in the range from 1 through 32. Issue ?UIDSTAT to obtain the unique TID.

Output

AC0	Unchanged (or error code)
AC1	Unchanged
AC2	Unchanged

Error Codes in AC0

ERPRH	Specified PID not in process hierarchy
ERSGO	Signal already outstanding
ERTID	Task does not exist

Why Use It?

You can use ?SIGNL, along with ?WTSIG and ?SIGWT, to allow tasks to synchronize access to global memory that identical local servers share.

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

?SIGNL is similar to the IPC system call ?ISEND. However, unlike ?ISEND, ?SIGNL does not cause any data movement. Instead, ?SIGNL allows a task to signal one of the following:

- Another task in the same process.
- Another task in a different process.

A task can issue ?SIGNL, even though the target task is not waiting to receive a signal. In this case, the operating system remembers the target. Then, when the target issues a ?WTSIG system call, it does not wait. Also, a task can issue ?SIGNL from interrupt level.

Warnings

The fast interprocess signaling mechanism is used by Data General supplied software, including (but not limited to) the PMGR interface routines found in the Agent in Ring 3 of all processes in the

operating system. And, the system makes absolutely no checks as to the legitimacy of a ?SIGNL issued by anyone or to anyone.

These two warnings have the following implications.

If a task ?SIGNLs another task, it must be absolutely sure that the next ?SIGWT or ?WTSIG issued by the receiving task will be the one it expects to receive the ?SIGNL. If such is not the case, the ?SIGNL may go to a ?SIGWT or ?WTSIG issued by another routine used by the task elsewhere in the process (e.g., the PMGR interface in Ring 3, or the INFOS local server in Ring 4). There is no addressing of the message as is provided by the IPC message port number or the ?REC/?XMT mailbox location.

Second, after any task resumes execution after receipt of a ?SIGNL by ?WTSIG or ?SIGWT, it is incumbent upon the receiver to verify that the condition being waited for has in fact occurred. The operating system does not authenticate ?SIGNLs sent.

Notes

- See the descriptions of ?SIGWT and ?WTSIG in this chapter.

?SIGWT Signals another task and then waits for a signal.

?SIGWT
error return
normal return

Input

AC0	Reserved (Set to 0.)
AC1	PID of the target process (The caller can specify its own PID; however, the OS validates whatever PID is specified.)
AC2	Unique TID of the target task in the range from 1 through 32. Issue ?UIDSTAT to obtain the unique TID.

Output

AC0	Unchanged (or error code)
AC1	Unchanged
AC2	Unchanged

Error Codes in AC0

ERPRH	Specified PID not in process hierarchy
ERSCI	System call at interrupt level
ERSGO	Signal already outstanding

Why Use It?

You can use ?SIGWT, along with ?WTSIG and ?SIGNL, to allow tasks to synchronize access to global memory that identical local servers share.

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

?SIGWT is similar to the IPC system call ?IS.R. However, unlike ?IS.R, ?SIGWT does not cause data movement. Instead, ?SIGWT allows a task to signal and then wait for a signal from one of the following:

- Another task in the same process.
- Another task in a different process.

A task can issue ?SIGWT, even though the target task or process is not waiting to receive a signal. In this case, the operating system remembers the target. Then, when the target issues a ?WTSIG system call, it does not pend. ?SIGWT will wait for a signal from another task (if one has not already been issued) before it wakes up.

Warnings

The fast interprocess signaling mechanism is used by Data General supplied software, including (but not limited to) the PMGR interface routines found in the Agent in Ring 3 of all processes in the operating system. And, the system makes absolutely no checks as to the legitimacy of a ?SIGNL issued by anyone or to anyone.

These two warnings have the following implications.

If a task ?SIGNLs another task, it must be absolutely sure that the next ?SIGWT or ?WTSIG issued by the receiving task will be the one it expects to receive the ?SIGNL. If such is not the case, the ?SIGNL may go to a ?SIGWT or ?WTSIG issued by another routine used by the task elsewhere in the process (e.g., the PMGR interface in Ring 3, or the INFOS local server in Ring 4). There is no addressing of the message as is provided by the IPC message port number or the ?REC/?XMT mailbox location.

Second, after any task resumes execution after receipt of a ?SIGNL by ?WTSIG or ?SIGWT, it is incumbent upon the receiver to verify that the condition being waited for has in fact occurred. The operating system does not authenticate ?SIGNLs sent.

Notes

- See the descriptions of ?SIGNL and ?WTSIG in this chapter.

?SINFO — Gets selected information about the current operating system.

?SINFO
error return
normal return

Operating System Differences	
Accumulator	
Input and Output	None
Error Codes	None
Parameter Packet	Some

Input

AC0	Reserved (Set to 0.)
AC1	Reserved (Set to 0.)
AC2	Word pointer to the ?SINFO packet

Output

AC0	Undefined
AC1	Undefined
AC2	Undefined

Error Codes in AC0

ERMPR	System call parameter address error
ERPRE	Invalid system call parameter
ERVBP	Invalid byte pointer passed as a system call argument
ERVWP	Invalid word pointer passed as a system call argument

Why Use It?

?SINFO is a way to find out your system's revision number, memory configuration, and current executing operating system name. If an extended call is made to return the operating system name, and the executing system is installed on the master LDU, then ?SINFO will return INSTALLED SYSTEM as the 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

?SINFO passes information about the version of the operating system that you are running to the packet that you specify in AC2. Figure 2–219 shows the structure of the ?SINFO packet.

Note that the packet specifications you supply are the byte pointers to two buffer areas: one area to receive the name of the master logical disk unit (LDU), and another area to receive the system identifier. Also, if the pathname to the currently executing operating system is desired, the packet

offset ?SIRS must be set equal to ?PKR1, and the byte pointer to the buffer to hold the name must be included. You must reserve an area in your logical address space ?SIPL words long for the ?SINFO packet if the pathname of the executing operating system is not wanted, or else ?SIEX words long if the name is wanted.

Figure 2–219 shows the structure of the ?SINFO packet.

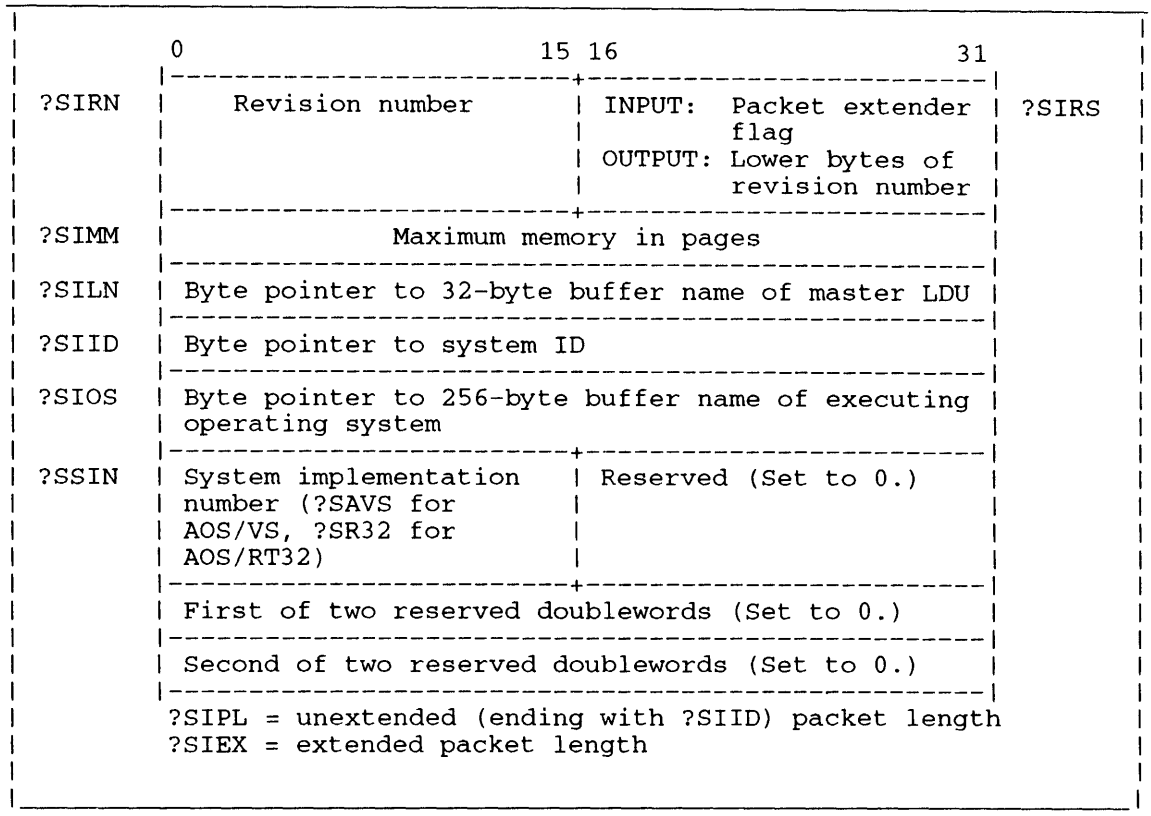


Figure 2–219. Structure of ?SINFO Packet

The operating system returns its major and minor revision numbers as octal values to the left and right byte of offset ?SIRN respectively. If the revision number is 00.00.01.01, for example, ?SIRN contains 00101. The operating system returns 2100 in offset ?SIRN for AOS/VS II Release 1.00.

Offset ?SIMM contains the number of the highest memory page. In a 256–Mbyte system with contiguous memory, for example, the value of ?SIMM is octal 400000; that is decimal 131072. The first memory page is page zero, and the last is the highest memory page known to the operating system. On some systems, not all memory is contiguous and available to the operating system. Thus memory pages may not be contiguous up to the page value returned in offset ?SIMM. Also, on some systems part of system memory is dedicated to firmware scratch space.

If you want the operating system to return the name of the master LDU you are using and the system identifier, set up two 32–byte receiving areas in your logical address space. Use offset ?SILN as a byte pointer to the buffer area for the master LDU name, and use offset ?SIID as a byte pointer to the buffer for the system identifier. If you do not want the name of the master LDU or the name of the system identifier, place 0 in the appropriate offset(s) and omit the corresponding buffer area(s).

?SINFO Continued

If you want the operating system to return the pathname of the executing operating system, then make an extended ?SINFO call. The ?SIRS word in the packet is used to indicate an extended call on INPUT, and will contain the second 2 bytes of the system Revision Number on OUTPUT. If this word is set to ?PKR1 a new doubleword byte pointer has been appended to the packet. This byte pointer is to the ?MXPL–byte buffer which will hold the pathname. If this word is zero, an unextended ?SINFO call is being made, and the packet does not contain the pathname byte pointer.

If either the word pointer in AC2 or the byte pointers in the packet refer to locations that are outside your logical address space, ?SINFO fails and the system returns the error code ERMPR in AC0. If ?SIRS is not set to either 0 or ?PKR1, an "Invalid system call parameter" error will result. If an extended call is made, but the byte pointer is invalid, an "Invalid byte pointer" error will result.

Sample Packet

```
PKT:    .BLK    ?SIEX           ;Allocate enough space for
                                ;the extended packet. Packet
                                ;length = ?SIEX.
        .LOC    PKT+?SIRN       ;Revision number.
        .WORD   0               ;The OS returns this value.
        .LOC    PKT+?SIRS       ;Supply ?PKR1 on input to specify
        .WORD   ?PKR1           ;an extended ?SINFO call.
        .LOC    PKT+?SIMM       ;Maximum memory.
        .DWORD  0               ;The OS returns this value.
        .LOC    PKT+?SILN       ;Byte pointer to buffer
                                ;that contains name of
                                ;master LD.
        .DWORD  MLD*2           ;Buffer is MLD.
        .LOC    PKT+?SIID       ;Byte pointer to system
                                ;ID buffer.
        .DWORD  SID*2           ;Buffer is SID.
        .LOC    PKT+?SIOS       ;Byte pointer to buffer
                                ;that contains name of
                                ;executing op sys.
        .DWORD  SOS*2           ;Buffer is SOS.
        .LOC    PKT+?SSIN       ;OS identifying number.
        .WORD   0               ;The OS returns this value.
        .LOC    PKT+?SIEX       ;End of packet.
MLD:    .BLK    16.
SID:    .BLK    16.
SOS:    .BLK    128.
```

?SLIST — Sets the search list for the calling process.

?SLIST
error return
normal return

Input

AC0	Reserved (Set to 0.)
AC1	Reserved (Set to 0.)
AC2	Byte pointer to the new search list

Output

AC0	Undefined
AC1	Undefined
AC2	Unchanged

Error Codes in AC0

ERFDE	File does not exist
ERSRE	Search list resolution error (There are too many directories in the search list.)

Why Use It?

You can access directories or files outside your own file structure by setting a search list.

Who Can Use It?

There are no special process privileges needed to issue this call. However, you need Execute access to the target file's parent directory.

What It Does

?SLIST sets the search list for the calling process. The symbol ?MXPL represents the maximum length for a search list. A search list cannot contain more than eight pathnames. (Input pathnames that resolve to the same directory count as one complete pathname.)

Use the following format for the search list string:

pathname<terminator>[pathname<terminator>...]<null>

where

terminator is either a null <000> or a New Line<012>.

You must conclude the string with a final null; e.g.,

:UDD:LISA:SPECS<012>:UTIL<000>^MACROS<012><000>

The operating system immediately resolves each pathname, that is, replaces it with a complete pathname from the root. The final search list, after resolution, can contain more than ?MXPL bytes. (This means that you may not be able to acquire an entire pathname with the ?GLIST system call.)

Note that any pathname prefixes you use in the string are relative to the current working directory.

Notes

- See the description of ?GLIST in this chapter.

?SONS — Gets a list of son processes for a target PID.

?SONS *[packet address]*
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 for the current process
AC1	One of the following: • –1 if AC0 contains a byte • 0 if AC0 contains a PID • ignored if AC0 contains –1
AC2	Address of the ?SONS packet, unless you specify the address as an argument to ?SONS

Output

AC0	Unchanged
AC1	Unchanged pointer
AC2	Address of the ?SONS packet

Error Codes in AC0

ERPRE	Invalid system call parameter
ERPRH	PID in AC0 points to a nonexistent process
ERPVS	Packet version not supported
ERRVN	Reserved value not zero
ERVBP	Invalid byte pointer passed as system call argument
ERVWP	Invalid address passed as system call argument

Why Use It?

Use this system call to obtain the PIDs of all son processes of a specified 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

?SONS returns all PIDs (or at least as many as will fit in the buffer provided) that are sons of a specified process. Thus, you have a determination of a process's hierarchy because you know its subordinate processes. The returned PIDs come from the entire range of PIDS on the system.

Figure 2–220 shows the structure of ?SONS's parameter packet, and Table 2–192 describes its contents.

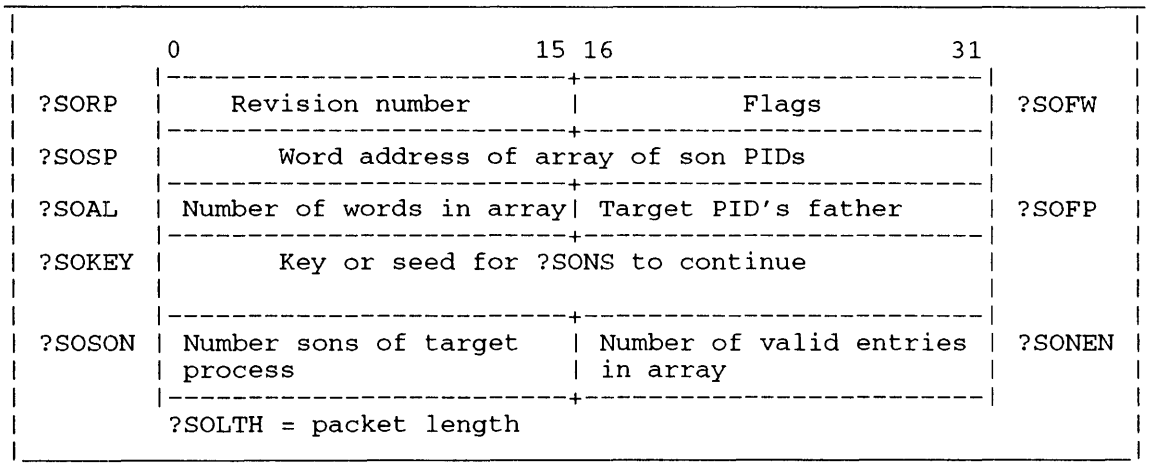


Figure 2–220. Structure of ?SONS Packet

Table 2–192. Contents of ?SONS Packet

Offset	Contents
?SORP	Packet revision number. Place zero here.
?SOFW	Flags. Set all unused bits to zero. Since no bit offsets are currently defined, place zero here.
?SOSP (double-word)	Word address of an array in which the OS will return the list of the PIDs of the sons of the target process. Each entry in the array is one word wide.
?SOAL	Length in words of the array that receives the list of son PIDs. ?SONS will not return more PIDs than the array will hold. You must place a nonnegative number here.
?SOFP	The operating system returns the PID of the process that is the father of the target process.
?SOKEY (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 of the target process, you must reissue the call to obtain the remaining PIDs. In this case, the prior issue of ?SONS has placed a number different from 0 in this offset; leave it here for all reissues. Don't keep the returned key number indefinitely, because it only is a snapshot of the current process hierarchy. Since this hierarchy typically changes frequently, the key probably isn't valid after significant processing time has elapsed.
?SOSON	The OS returns the number of PIDs of processes that were sons of the target process when ?SONS executed. You can use this number to adjust the size of the array. The OS always returns ?SOSON, even if ?SOAL contains 0.
?SONEN	The OS returns the number of valid and nonzero PIDs that it stored in the array. A program that uses this number doesn't have to count the number of nonzero entries to determine how many son PIDs were returned.

?SOPEN — Opens a file for shared access.

?SOPEN
error return
normal return

Input

AC0	Byte pointer to the target file's pathname
AC1	One of the following: • Target file's channel number • –1 to allow the OS to assign a channel number. (This is the default.)
AC2	One of the following: • 0 to open this file for read–only purposes • Any other value to have the file's ACL determine access privileges

Output

AC0	Unchanged
AC1	Target file's channel number
AC2	Unchanged

Error Codes in AC0

ERILO	Illegal open (You tried to ?SOPEN a file whose element size is not 4 or a multiple of 4.)
ERVBP	Invalid byte pointer passed as a system call argument

File system error codes

Why Use It?

?SOPEN opens a file for shared access. In general, you would use ?SOPEN and the other shared–page system calls to enable several processes to share the same re–entrant code or the same data. This reduces disk I/O and conserves memory.

Who Can Use It?

There are no special process privileges needed to issue this call. To restrict the caller's access privileges to the file, load AC2 with 0. This allows the calling process read–only access, which prevents the caller from modifying the file, regardless of the file's ACL. If you load AC2 with a nonzero number, then the file's ACL determines the caller's access privileges.

?SOPEN Continued

What It Does

?SOPEN opens a file for shared access. For the shared–page mechanism to work correctly, the file you open must have an element size of 4 or a multiple of 4.

If AC2 specifies both read and write, the operating system flushes the modified contents of the file to disk as soon as the caller issues ?SCLOSE to close the file.

Notes

- See the descriptions of ?SCLOSE, ?OPEN, and ?GCLOSE in this chapter.

?SOPPF
error return
normal return

Input

AC0	One of the following: • Byte pointer to the filename on first open • 0 on subsequent opens
AC1	One of the following: • Channel number • –1, to indicate that you want the OS to define the channel number
AC2	Packet address

Output

AC0	Unchanged
AC1	Channel number
AC2	Unchanged

Error Codes in AC0

EREO1	File is open, can't exclusively open
EFFAD	File access denied

Why Use It?

?SOPPF is the system call that opens a protected shared file.

Who Can Use It?

There are no special process privileges needed to issue this call. The restrictions concerning file access appear under the explanations of the ?PFFO flag.

What It Does

?SOPPF allows you to open a shared file in a protected manner. Then, the caller can issue shared–page system calls, just as if the channel had been opened with ?SOPEN. To close a shared file that has been opened with ?SOPPF, issue ?SCLOSE, just as if it had been opened with ?SOPEN.

When the ?PFFO flag is set in offset ?PFFLG, ?SOPPF allows a segment image to open a shared file in such a way that no other segment can open the file without its explicit permission. When the ?PFFO flag is not set in offset ?PFFLG, ?SOPPF allows a segment image to open a protected shared file using the access privileges granted by the first opener of the file.

To use ?SOPPF, you must supply a packet. Figure 2–221 shows the structure and Table 2–193 describes the contents of the ?SOPPF packet.

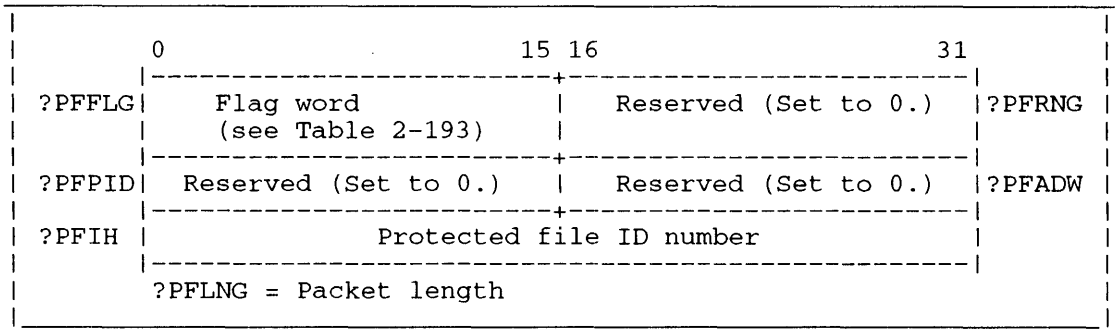


Figure 2–221. Structure of ?SOPPF Packet

Table 2–193. Contents of ?SOPPF Packet*

Offset	Meaning
?PFFLG	Flag word. ?PFFO set -- First open. ?PFFO not set -- Subsequent open. ?PFRW set -- Open for read/write. ?PFRW not set -- Open for read only.
?PFRNG	Reserved. (Set to 0.)
?PFPID	Reserved. (Set to 0.)
?PFADW	Reserved. (Set to 0.)
?PFIH (doubleword)	Protected file ID number.

* There is no default unless otherwise specified.

If you set the ?PFFO flag in offset ?PFFLG, then the caller must be the first to open the file. In this case, ?SOPPF causes the following to occur:

- The caller (user) passes a byte pointer to a pathname as input.
- If the file is already open, the operating system returns the error code EREO1.
- The operating system returns an ID number that uniquely identifies the protected shared file.
- The operating system returns a channel number for use by the caller.
- The operating system grants the caller whatever access privileges are indicated by the shared file's ACL.
- The operating system prevents the shared file from being opened, except by another ?SOPPF.

This first opener uses the ?PMTPF system call to permit other segment images to open the file. Only the first opener of a protected shared file can issue a ?PMTPF against the shared file. (See the description of ?PMTPF in this chapter for more information about access privileges.)

If the ?PFFO flag in offset ?PFFLG is not set, ?SOPPF causes the following to occur:

- The caller (user) passes the unique protected file ID as input.
- The operating system returns an error if the caller does not have explicit permission from the first opener of the shared file to open it.
- The operating system returns a channel number for the caller to use.
- The operating system grants the caller those access privileges that the first opener's ?PMTPF specified.

Notes

- See the description of ?PMTPF in this chapter.

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

Input

AC0	Reserved (Set to 0.)
AC1	Target file's channel number (returned to AC1 as output to ?SOPEN)
AC2	Address of the ?SPAGE packet, unless you specify the address as an argument to ?SPAGE

Output

AC0	Undefined
AC1	Unchanged
AC2	Address of the ?SPAGE 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
ER_FS_DIRECTORY_NOT_AVAILABLE
Directory not available because the LDU was force released (AOS/VS II only)

Why Use It?

You can use ?SPAGE to move a page or pages into the shared area of your logical address space. Offset ?PSTI in the ?SPAGE packet allows you to restrict the calling process to read access only. This is a way of preventing modifications to shared routines or shared data.

Who Can Use It?

There are no special process privileges needed to issue this call, and there are no restrictions concerning file access. Set Bit 0 of offset ?PSTI to limit access to the file to Read access only.

What It Does

?SPAGE validates a shared page or pages in the caller's logical address space. Note that you must open the target file with ?SOPEN before you can issue ?SPAGEs against it.

If the target page is already on the LRU chain and, therefore, memory resident, the operating system checks the caller's access to it, and adds the page to the working set when the caller references it. If the page is not in memory, the operating system reads it into memory before it adds the page to the working set (at the time the page is referenced).

Specify the number of blocks you want to read or write in the right 15 bits of offset ?PSTI.

Figure 2–222 shows the structure of the ?SPAGE packet, and Table 2–194 describes its contents. Note that the packet is structurally identical to the packet for the block I/O calls ?RDB and ?WRB, which are described in this chapter.

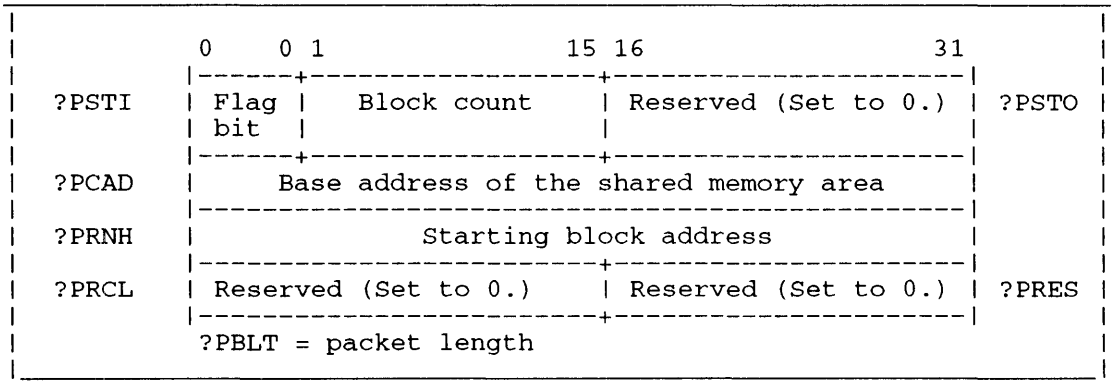


Figure 2–222. Structure of ?SPAGE Packet

Table 2–194. Contents of ?SPAGE Packet*

Offset	Contents
?PSTI	Bit 0 contains the following flag: ?SPRO -- allow Read access only. Set to 1 if you want to write protect the page(s). (This will even write protect pages opened with ?SOPEN.) Right 15 bits contain the block count. The block count is the number of disk blocks to be transferred into the shared area. This number must be 4 or a multiple of 4.
?PSTO	Reserved. (Set to 0.)
?PCAD (doubleword)	Starting address in memory; that is, the base address of the shared memory area to use. (This is always the start of a page boundary.)
?PRNH (doubleword)	Address of the file's starting block.
?PRCL	Reserved. (Set to 0.)
?PRES	Reserved. (Set to 0.)

* There are no defaults unless otherwise specified.

?SPAGE Continued

Sample Packet

```
PKT:    .BLK    ?PBLT           ;Allocate enough space for the
                                ;packet.  Packet length = ?PBLT.
        .LOC    PKT+?PSTI       ;Number of disk blocks to read into
        .WORD   4               ;shared area (must be 4 or a multiple
                                ;of 4).  Transfer four blocks.
        .LOC    PKT+?PSTO       ;Reserved.
        .WORD   0               ;You must set this value to 0.
        .LOC    PKT+?PCAD       ;Base address of shared memory area.
        .DWORD  BUFFER          ;BUFFER is address of data buffer.
        .LOC    PKT+?PRNH       ;Starting block address.
        .DWORD  0               ;Block 0 or first block of file.
        .LOC    PKT+?PRCL       ;Reserved.
        .WORD   0               ;You must set this value to 0.
        .LOC    PKT+?PRES       ;Reserved.
        .WORD   0               ;You must set this value to 0.
        .LOC    PKT+?PBLT       ;End of packet.
```

Notes

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

?SPOS — Sets the position of the file pointer.

?SPOS
error return
normal return

Input

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

Output

AC0	Undefined
AC1	Undefined
AC2	Address of the ?READ or ?WRITE packet

Error Codes in AC0

ERBOF	Positioning before beginning of the file
EREOF	End of file
ERFNO	Channel not open
ERICN	Illegal channel
ERVWP	Invalid word pointer passed as a system call argument

Why Use It?

You can use ?SPOS to change the file pointer's position in a file without doing I/O.

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

?SPOS repositions the file pointer of a file that is currently open for I/O (an open file or a file you are currently reading or writing). The operating system maintains a separate file pointer for each open channel to keep track of I/O performed across the channel.

By default, the file pointer moves forward sequentially as the operating system executes each read or write. You can position the file pointer before an I/O sequence by manipulating the file pointer specifications in the I/O packet. Also, you can issue ?SPOS to change the position of the file pointer without performing I/O. The file pointer parameters are

- ?IRNH, which is a double–precision integer that indicates the position of the file pointer.
- ?IRCL, which states the maximum number of bytes to read or write.
- Bit ?IPST in offset ?ISTI, which determines whether the value in ?IRNH is an absolute or a relative value.

You manipulate these parameters with ?SPOS just as you would with ?READ or ?WRITE. Table 2–195 summarizes the file–pointer settings. Note that you cannot issue ?SPOS against a labeled magnetic tape file.

Table 2–195. File–Pointer Settings

?IPST Value	?IRNH Value	Pointer Position
0	0	At the next sequential record in the file (relative to its current position).
0	n	For fixed-length records, at relative record offset n (forward or backward from the current file pointer position); for all other record types, at relative byte offset n.
1	0	At the beginning of the file.
1	-1	At the end of the file.
1	n (any integer)	For fixed-length records, at the nth record; for all other types, at the nth byte. (The first record or byte is 0.)

If the record type is fixed–length, set offset ?IRCL to the record length or to –1, the default value. Use the default only if you specified the record length in the ?OPEN, ?READ, or ?WRITE packet. For all other record types, set ?IRCL to 0. Set bit ?IPST to 1 if you want the operating system to interpret ?IRNH as an absolute value. Set ?IPST to 0 if you want the operating system to interpret the file pointer position as a relative value.

See the description of ?READ/?WRITE in this chapter for more information on setting the file pointer's position.

Notes

- See the descriptions of ?OPEN, ?READ, and ?WRITE in this chapter.

?SRCV Receives data or a control sequence over a BSC line.

?SRCV
error return
normal return

Input

AC0	(Used by multipoint control stations only) One of the following: • −1 for a general poll • Byte pointer to a poll address for a specific poll
AC1	Channel number assigned to the line
AC2	Address of the ?SRCV packet

Output

AC0	Unchanged
AC1	Unchanged
AC2	Unchanged

Error Codes in AC0

ERCRC	CRC check (Indicates a cyclical redundancy block check error.)
ERDIS	Disconnect occurred on a switched line
ERENQ	ENQ received after time−out (The OS detected an ENQ from the sending station after it reached the limit for retries.)
EREOT	EOT character received
EREPL	End of polling list reached
ERISE	Input status error
ERLIS	Line in session (You tried to issue two receive initial system calls in a row.)
ERLNA	Attempt to enable nonsynchronous line
ERNAK	Transmission failure (The OS received a NAK while it was waiting for a line bid, and it has reached its limit for retries (point−to−point lines only).)
ERNPL	Polling list not defined (This error occurs only on multipoint lines.)
ERNSL	Device associated with the channel number is not a synchronous line
ERSCS	You tried to issue a receive continue without a prior receive initial
ERSIM	There is an outstanding system call on this line
ERTOF	Time−out value exceeded
ERTRF	Transmitter failure
ERUNI	Unknown or inappropriate response received
ERVBP	Invalid byte pointer passed as a system call argument

Why Use It?

You can use ?SRCV whenever you want to receive data over an enabled BSC line.

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

?SRCV prepared the calling station to receive a block of data or data–link control characters over an enabled BSC line. Before you issue ?SRCV, perform the following steps:

1. Reserve a receive buffer in your logical address space for the incoming data or control sequence.
2. Set up the ?SRCV packet in your logical address space.
3. Load AC1 with the BSC line's channel number.
4. Load AC2 with the packet address.

Figure 2–223 shows the structure of the ?SRCV packet, and Table 2–196 describes each offset.

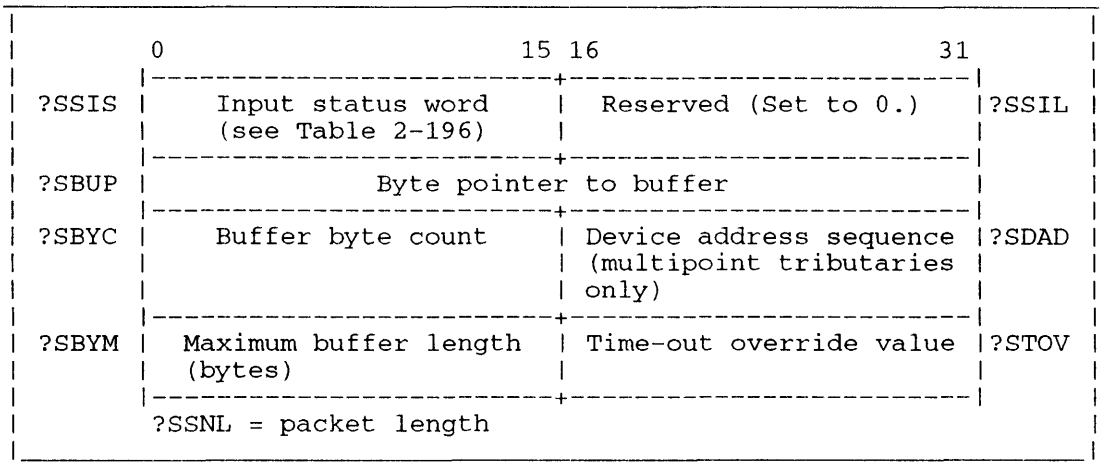


Figure 2–223. Structure of ?SRCV Packet

Table 2–196. Contents of ?SRCV Packet*

Offset	Contents	Input Value	Output Value
?SSIS	Input status word:		
	Text mode.	None.	?TRAN--If received transparent data.
	Block start.	None.	?STXB--STX received. ?SOHB--SOH received.
	Intermediate text block indicator.	None.	?SITB.
	Multipoint information.	None.	?SPLR--Your system was polled. ?SSLR--Your system was selected.
	Station ID indicator	?SRID--Receive a station's ID or send yours.	
	Response type (receive continue only)		
	(Directs OS to respond to the sender with the specified control character.)	?SRVI--Reverse interrupt. ?SNAK--NAK. ?SACK--ACK0 or ACK1. ?SAK0--Override ACK1, send ACK0. ?SAK1--Override ACK0, send ACK1.	Unchanged.
	Call type.	?CINT--Receive initial. ?CONT--Receive continue.	If ?CINT was set and the bid for the line was succesful, the call type is changed to ?CONT.

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

Table 2–196. Contents of ?SRCV Packet*

Offset	Contents	Input Value	Output Value
?SSIS (cont.)	Call type. (cont.)	The following apply to receive continues from multipoint tributaries only: ?SWAK--Send WACK; then wait for poll or select. ?SPET--Send EOT; then wait for poll or select. ?SPRV--Send RVI; then wait for poll or select. ?SPNK--Send NAK; then wait for poll or select.	
	Component Selection.	None.	?SDAC--Received device address characters.
?SSIL	Reserved.	Set to 0.	N/A.
?SBUP (double word)	Buffer byte pointer.	Byte pointer to a receive buffer for the data.	Unchanged.
?SBYC	Buffer byte count.	None.	Number of bytes received.

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

Table 2–196. Contents of ?SRCV Packet*

Offset	Contents	Input Value	Output Value
?SDAD	Device address sequence.	None.	Left-justified device address, 2-character maximum. (You receive the device address immediately after your poll or select address.) The OS returns this value to a multipoint tributary only after an ?SRCV initial call.
?SBYM	Maximum buffer length.	Byte length of receive buffer.	Unchanged.
?STOV	Time-out override value in seconds. DEFAULT = -1 (8 seconds)	Length of time the OS will wait for the remote station to respond, before it enters its BSC recovery procedure.	Unchanged.

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

Set Bit ?SRID in offset ?SSIS when you expect to receive an ID bid or when you are replying to an ID bid by sending

<your station's ID>ACK or <your station's ID>NAK.

If you receive an ID bid, the ID will be in the buffer and its length will be in offset ?SBYC. See the section "Station Identification" in the description of system call ?SEBL.

There are two types of ?SRCV system calls:

- Receive initial calls, which open the transmission.
- Receive continue calls.

The operating system executes these two types of ?SRCVs differently, depending on whether the caller is a station on a point–to–point line, or the control station or a tributary on a multipoint line. The following discussion is divided to reflect these differences.

Note that you must set certain parameters regardless of the kind of station you are calling from. The next section describes these parameters.

Required Input

Set offset ?SBUP to the address of the receive buffer you set aside for the data, and set offset ?SBYM to the length of the receive buffer.

In offset ?SSIS, the input status word, specify the type of receive you want to perform. The ?CINT mask defines the ?SRCV as a receive initial call, and the ?CONT mask defines it as a receive continue. Be sure to set the correct mask. You cannot issue two receive initial calls in a row, nor can you issue a receive continue before you issue a receive initial call.

If you set ?CINT and the bid for the line is successful, the call type field is changed to ?CONT. As a result, should you get an error, you can check whether the error occurred during the bid for the line or during reception of the data block.

If you are using the user identification function, then the call type field ?CINT is changed to ?CONT after the second initial receive call.

Receive Continue Calls

For receive continue calls, you must set one of the response–type masks in offset ?SSIS. The response–type field defines the control character the operating system uses to reply to the sending station after each transmission. The response–type options are

- Return a negative acknowledgment (?SNAK) to the sender.
- Return a positive acknowledgment (ACK0 or ACK1) in the correct alternating sequence (?SACK) to the sender.
- Return an ACK0 positive acknowledgment (?SAK0) to the sender.
- Return an ACK1 positive acknowledgment (?SAK1) to the sender.
- Return a reverse interrupt (?SRVI) to the sender.

Note that there are three variations for a positive acknowledgment. Under standard BSC protocol, the operating system alternates the ACK0 and ACK1 characters; that is, if a station uses ACK0 as its first positive acknowledgment, it will use ACK1 as its second positive acknowledgment, and vice versa. The ?SAK0 and ?SAK1 options let you specify a single positive acknowledgment character for the duration of the ?SRCV.

Response type ?SRVI directs the operating system to reply with a reverse interrupt (RVI) character. Use this response type if you want to interrupt the transmission to pass the sending station a high–priority message.

You can continue to issue receive continue calls until you have obtained all the sending station's data. You will know the transmission has ended when the operating system returns error code EREOT in AC0.

If you issue a receive continue call and the sending station fails to respond or responds inappropriately, the operating system enters its BSC error–recovery procedures.

?SRCV from a Point–to–Point Station

If you issue ?SRCV as a receive initial call from a point–to–point station, the operating system waits for a bid sequence from the other station before it executes ?SRCV. By default, the operating system waits 8 seconds. You can set a different time–out value in offset ?STOV.

If the other station fails to send a bid sequence in time, the operating system enters the BSC error–recovery procedure for receive initial calls. If the operating system receives the other station's bid in time, the ?SRCV succeeds, and the operating system returns the first block to your receive buffer.

?SRCV from a Multipoint Control Station

When you issue ?SRCV from a control station on a multipoint line, the operating system performs either a general poll to solicit data from all your enabled tributaries or a specific poll to solicit data from one tributary. Your input to AC0, before you issue ?SRCV, determines what kind of polling the operating system performs.

On a receive–initial general poll, the operating system sends each entry in the polling list down the BSC line, starting with the entry for the first relative terminal (tributary). If a tributary has no data to send, it responds to its poll address with an EOT character, and the operating system continues polling.

A general polling operation ends when one of the following occurs:

- A tributary sends data to the control station. (The operating system then places the data in the control station's receive buffer and loads AC1 with the sender's relative terminal number.)
- The operating system reaches the end of the polling list without receiving data. (The operating system returns error code EREPL in AC0.)
- A tributary fails to respond within the time value that you specify in packet offset ?STOV. (The operating system returns error code ERTOF in AC0), or a tributary responds inappropriately. The operating system returns error code ERUNI in AC0.)

Receive continue calls from multipoint stations work the same as receive continue calls from point–to–point stations. (See the "Receive Continue Calls" section for details.)

?SRCV from a Multipoint Tributary

An ?SRCV receive initial call issued from a tributary signals the operating system that the tributary is prepared to accept data from the control station. The operating system responds by monitoring the line for your tributary's poll or select address. If the control station fails to send the poll or select address within the interval you set in ?STOV, ?SRCV fails and the operating system returns error code ERTOF in AC0.

If your tributary was polled in time, the operating system sets bit ?SPLR in offset ?SSIS and takes the normal return from the ?SRCV. You should respond by issuing an ?SSND send continue call if you have data to send, or by issuing an ?SSND EOT (end–of–transmission) if you do not have data to send.

If your tributary was selected in time, the operating system sets bit ?SSLR in offset ?SSIS and takes the normal return. If you are ready to receive data, respond to the select sequence with an ?SRCV receive continue that has one of the positive response bits set (?SACK, ?SAK0, or ?SAK1). ?SAK0 and ?SAK1 override the operating system's usual alternation between ACK0 and ACK1, and select a specific positive response (ACK0 or ACK1) for this receive call.

These parameters also reset the operating system's alternation pattern. For example, if you select ?SAK0, the next positive response will be ACK1, unless you override the alternation pattern again.

If you are not ready to receive data, issue a ?SSND continue with the call type set to ?SWAK (wait–before–transmission, positive acknowledgment).

?SRCV Output Values

A normal return from ?SRCV causes the operating system to set certain masks in offset ?SSIS. These masks indicate the text mode of the transmission and the presence of several data–link control characters or poll, select, or device addresses. The previous section describes two of these masks, ?SPLR and ?SSLR. Table 2–197 lists all of the masks and describes their meanings.

Table 2–197. Masks Returned on ?SRCV System Calls

Mask	Information Returned	Meaning
?TRAN	Text mode.	If set, transparent text received.
?SOHB		If ?SOHB is set, start of header (SOH) character received.
?STXB	Start of block.	If ?STXB is set, start of text (STX) character received.
?ETBB		If ?ETBB is set, start of text (STX) character received.
?ETXB	End of block.	If ?ETXB is set, end of text (ETX) character received.
?SITB	ITB received.	If set, end of intermediate transmission block (ITB) character received.
?SPLR	Poll indicator.	If ?SPLR is set, poll address received.
?SSLR	Select indicator.	If ?SSLR is set, select address received.
?SDAC	Device/component selector.	If set, device address received.
?CONT	Call type.	If set, ?CINT resulted in successful bid for the line.

The operating system sets the ?SITB flag when one or more ITB (intermediate text block) characters delimit certain portions of the text block. When a block contains these characters, the operating system precedes each intermediate text block portion with a value that indicates its length. Figure 2–224 shows a receive buffer that consists of three intermediate text block portions with lengths of 4, 5, and 6 characters, respectively.

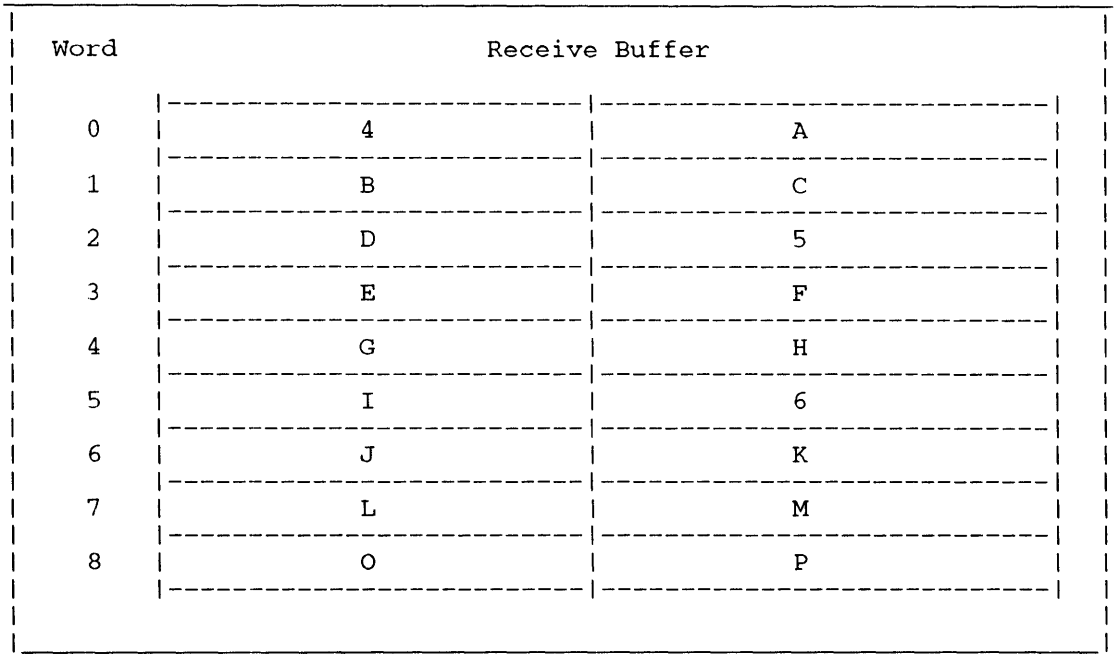


Figure 2–224. ITB Receive Buffer Format

Notes

- See the description of ?SSND in this chapter.

?SSHPT
error return
normal return

Input

AC0	Logical page number of the first page in the shared memory area (range: 1 through 262,143)
AC1	Number of pages in the new shared area
AC2	Reserved (Set to 0.)

Output

AC0	Unchanged
AC1	Unchanged
AC2	Undefined

Error Codes in AC0

ERMEM	Insufficient memory available (You may have tried to overlap the shared area into the unshared area.)
ERSHP	Error on shared partition set

Why Use It?

You might issue ?SSHPT before you issue ?SOPEN and ?SPAGE. This would expand the shared area of your logical address space before calling in one or more shared pages. A companion call, ?GSHPT, returns the current size of the shared 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

?SSHPT either increases or decreases the caller's valid shared–page partition. Pages can only be shared within an established boundary. ?SSHPT effectively adds or removes space for potential shared pages.

As the error messages suggest, the current shared and unshared boundaries are important factors in the success of an ?SSHPT. The operating system takes the error return and passes error codes to AC0 under the following conditions:

- When you try to overlap the shared area of the logical address space with the unshared area (error code ERMEM).
- When you use ?SSHPT to transfer unshared pages to the shared area (error code ERSHP).
- When you try to release shared pages you are currently using (error code ERSHP).

Notes

- See the descriptions of ?SOPEN, ?SPAGE, and ?GSHPT in this chapter.

?SSID
error return
normal return

Input

AC0	Reserved (Set to 0.)
AC1	Reserved (Set to 0.)
AC2	Byte pointer to a 32–byte buffer that contains the ASCII string to use as the system identifier

Output

AC0	Undefined
AC1	Undefined
AC2	Unchanged

Error Codes in AC0

ERMPR	System call parameter address error
ERPRE	Invalid system call parameter
ERPRV	Caller not privileged to make this call
ERVBP	Invalid byte pointer passed as a system call argument

Why Use It?

Typically, operators set the system identifier for each computer in their installation. EXEC returns the system identifier as part of the message:

TYPE NEW–LINE TO BEGIN LOGGING ON

EXEC looks at the system identifier only once — when EXEC is created. Subsequent changes to the system identifier have no effect.

To find out what the system identifier is for your current system before you issue ?SSID, issue ?GSID.

Who Can Use It?

Only the operator process (PID 2), or a process that has System Manager privilege, can issue ?SSID. There are no restrictions concerning file access.

What It Does

?SSID assigns a unique system identifier to your current system.

Before you issue ?SSID, load AC2 with the address of the system identifier (defined elsewhere in your program). The system copies the string you specify into the system SYSID buffer. By convention, anything appearing after the first null (if any) in the system identifier is ignored by most processes.

Notes

- See the description of ?GSID in this chapter.

?SSND
error return
normal return

Input		Output	
AC0	Byte pointer to a tributary's select address and device address (used by multipoint control station, only)	AC0	Unchanged
AC1	Channel number assigned to the line	AC1	Unchanged
AC2	Address of the ?SSND packet	AC2	Unchanged

Error Codes in AC0

ERBNK	Bid error (too many NAKs) (The ?SSND caller received a NAK in response to its line bid (point–to–point), or its select sequence (multipoint control station).)
ERCNV	Conversational reply received
ERCRC	CRC check
ERCTN	Contention situation while bidding (Both stations on a point–to–point line tried to bid at the same time.)
ERDIS	Disconnect received on a switched line (This error also occurs when a ?SSND caller receives a DLE EOT in response to its TTD sequence.)
EREOT	EOT character received (The ?SSND caller received an end of transmission character in response to its data block or WACK character.)
ERLIS	Line in session on ?SSND (You tried to issue two ?SSND initial calls in a row.)
ERLNA	I/O request for disabled line
ERNAK	Transmission failure (NAK count) (The OS received a NAK in response to a data block, or a NAK WACK sequence from the receiver; this also occurs when the sending station receives a NAK in response to an ENQ (repeat previous response), and the retry limit is exceeded.)
ERNSL	Attempt to enable nonsynchronous line
ERRVI	RVI response received (An RVI character received in response to the sender's bid (point–to–point only) or select sequence (multipoint only), or following the receipt of a WACK.)
ERSCS	?SSND continue without line in session (You tried to issue an ?SSND continue without a previous ?SSND initial.)
ERSIM	There is an outstanding call on this line
ERTOF	Transmission failure (The time–out value has elapsed.)
ERTRF	Transmitter failure
ERUNI	Uninterpretable response received
ERVBP	Invalid byte pointer passed as a system call argument
ERWAK	A WACK character received in response to a line bid (point–to–point only) or a select sequence (multipoint only)

Why Use It?

?SSND allows you to send a block of data over an enabled BSC line.

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

?SSND lets the calling station send a block of data or data–link control characters over an enabled BSC line. A process can issue ?SSND to bid for the BSC line (send initial), to bid for the line and then send data (send continue), or to send data (send continue).

Before you issue ?SSND, set up a packet in your logical address space, load AC2 with its address, and load AC1 with the line's channel number. Figure 2–225 shows the structure of the ?SSND packet, and Table 2–198 describes each offset.

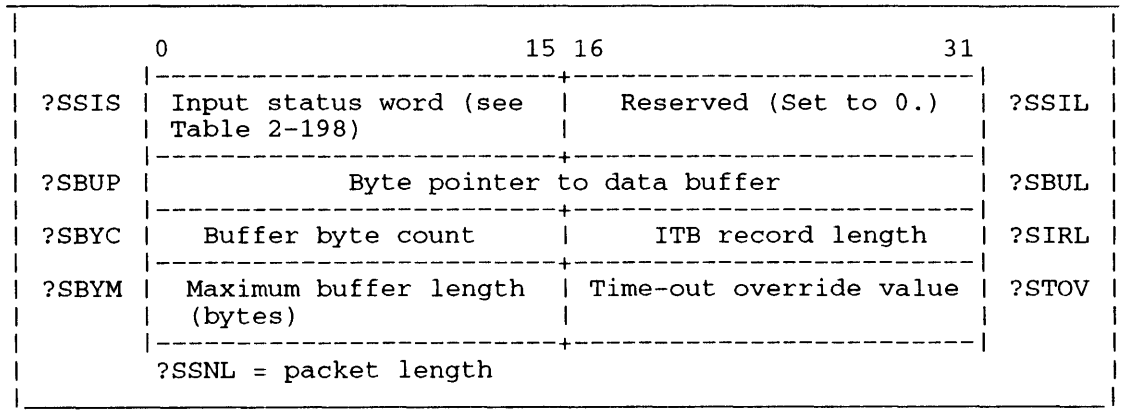


Figure 2–225. Structure of ?SSND Packet

Table 2–198. Contents of ?SSND Packet*

Offset	Contents
?SSIS	Input status word. Data Type. ?NTRN--Nontransparent Text Mode. ?TRAN--Transparent Text Mode. DEFAULT = ?NTRN (Nontransparent Text Mode). Block Start. ?STXB--Start of text. ?SOHB--Start of header. DEFAULT = ?STXB (start of text). Block Type. ?ETBB--End of block. ?ETXB--End of text. DEFAULT = ?ETBB (end of block). Send Intermediate Text Blocks. ?SITB. DEFAULT = No intermediate text blocks. Accept Conversational Replies. ?SCON. DEFAULT = Do not accept conversational replies. Send Device Address Characters (for select sequences only). ?SDAC. DEFAULT = Do not send device address.

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

Table 2–198. Contents of ?SSND Packet*

Offset	Contents
?SSIS (cont.)	Call type. ?CINT--send initial (with data). ?CONT--send continue (with data). ?SWAK--send WACK sequence. ?SEOT--send EOT sequence. ?SDET--send DLE ETB or DLE ETX sequence (to signal end of transparent text mode). ?STTD--send TTD sequence (to signal temporary text delay). ?SINT--send RVI. ?SDIS--perform logical disconnect. ?SNID--send your station's ID.
?SRES	Reserved. (Set to 0.)
?SBUP (doubleword)	Buffer byte pointer. This is a byte pointer to the buffer in which the data you are sending resides.
?SBYC	Buffer byte count. This is the number of bytes you wish to send.
?SIRL	ITB record length. This is the length (in bytes) of the intermediate text blocks, which are records delimited by an ITB character. The ITB record length should not be greater than 255. DEFAULT = 0 (no ITBs).
?SBYM	Maximum buffer length. This is the byte length of the data buffer.
?STOV	Time-out override value in seconds. This is the length of time the OS will wait for a reply before it begins the BSC error-recovery procedures. DEFAULT = -1 (8 seconds).

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

Input Status Word

Offset ?SSIS, the input status word, defines certain characteristics of the block you will send.

The data type field specifies whether the data is in Transparent Text Mode (?TRAN), or Nontransparent Text Mode (?NTRN). Under transparent text mode, most data–link control characters are treated only as bit patterns; that is, they have no control significance. The exceptions are DLE STX, which signals the beginning of Transparent Text Mode, and DLE ETB or DLE ETX, which signals the end of Transparent Text Mode.

Although the operating system inserts the necessary data–link characters, including the BCC character, when you issue ?SSND, three fields in offset ?SSIS let you specify the data–link characters you want to use.

The block start field specifies whether the operating system should preface the block with an STX (start of text) character or with an SOH (start of header) character. Set mask ?SOHB if you are sending header information in this transmission and set mask ?STXB if you are sending text. The ?STXB mask is the default.

The block type field directs the operating system to append either an ETB (end of text block) or an ETX (end of text) character to the data block. The ETX character terminates the last text block in a message. Thus, you should set mask ?ETXB if this is the last block you are sending. The default for this field is ?ETBB. This mask corresponds to the ETB character, which must terminate every text block except the last one.

The send intermediate text block field directs the operating system to delimit portions of the data block with intermediate transmission (ITB) characters. If you select this option (by setting mask ?SITB), you must set offset ?SIRL to the length of the intermediate text blocks. If you set ?SIRL to 4, for example, the operating system places an ITB character after every fourth byte in the text block. The operating system ignores the contents of ?SIRL if bit ?SITB is not set.

Select the ?SCON mask if you want to accept conversational replies from the receiving station. Conversation mode allows the receiver to respond to the block you send with header information or data, instead of a positive ACK0 or ACK1 sequence. Note that the block you send must be a complete data block (not a data–link sequence) for conversational mode to work. In general, you should not give a conversational reply to a header block or a block that ends with an ETB character.

The call type field defines ?SSND as one of the following types:

- ?CINT, which is a send initial with data. (If the bid for the line is successful, the operating system changes this to ?CONT.)
- ?CONT, which is a send continue with data.
- ?SWAK, which is a wait–before–transmission (positive acknowledgment).
- ?SEOT, which is an end–of–transmission.
- ?SDET, which is a line disconnect call.
- ?STTD, which is a temporary text delay.
- ?SNID, which allows you to send your station's ID. Place the ID in the data buffer, its number of bytes in offset ?SBYC, and the largest possible number of bytes in the explected reply ID in offset ?SBYM. When ?SSND finishes you must check offset ?SBYC to see if the receiving station replied with an ID. If so, it will be in the data buffer. See the section "Station Identification" in the description of system call ?SEBL.

Table 2–199 describes each type of call.

Other ?SSND Offsets

Offsets ?SBUP and ?SBUL point to the buffer you reserved in your logical address space for the data you will send. Note that the buffer byte count you supply in offset ?SBYC must be for the

entire contents of the data buffer. If the data consists of intermediate text blocks (defined by ?SITB and ?SIRL), the data buffer should contain an exact multiple of the intermediate block length you specified in offset ?SIRL. Otherwise, the last intermediate text block will be shorter than the others, because the operating system does not pad the last intermediate text block.

If you set both ?SBUP and ?SBYC to 0 on a send initial, the operating system interprets your send call as a bid for the line. If the bid succeeds, the operating system simply takes the ?SSND normal return, without sending data. If you set either of these parameters to 0, but not both, the operating system takes the ?SSND error return and returns one of the following error codes in AC0:

Error Code	**Input**
ERBPE	?SBUP = 0 ?SBYC = any nonzero value
ERBCT	?SBUP = any nonzero value ?SBYC = 0

When you select the conversational reply option (?SCON), the operating system stores the response data from the receiver in the data buffer defined by ?SBUP, and records the length of the response data in offset ?SBYC.

Table 2–199. ?SSND Call Types

Station Type	Call Type	Action
Point-to-point	Send initial (with data) (?CINT)	Send ENQ (line bid) to the receiving station, and wait for an ACK0 response. - If line bid successful, ?CINT is changed to ?CONT; otherwise, remains the same. - If no ACK0 within time-out interval (?STOV), then begin BSC error-recovery procedures. If ACK0 within the time-out interval, then ?SSND data block, wait for ACK1 response. - If no ACK1 within time-out interval, then begin BSC error-recovery procedures. If ACK1 within time-out interval, then take ?SSND normal return.
	Send continue (with data) (?CONT)	Send data to receiving station, and await correct alternating response (ACK0 or ACK1). - If no response within time-out interval, then begin BSC error-recovery procedures.
	Send continue (with data) (?CONT)	- If correct response, then take ?SSND normal return.
All types (point-to-point and multipoint)	WACK sequence	Send WACK to receiver, and wait for an ENQ reply. - If no ENQ within time-out interval, then begin BSC error-recovery procedures. - If ENQ within time-out interval, then take ?SSND normal return.

(continued)

Table 2-199. ?SSND Call Types

Station Type	Call Type	Action
	EOT sequence or DLE EOT (a switched line disconnect)	Send control sequence and take ?SSND normal return. Calling station can issue ?SSND initial, ?SRCV initial, or ?SDBL (disable line) call now.
All types (point-to-point and multipoint)	Send continue with temporary text delay (?STTD)	Send TTD sequence and wait for NAK reply. - If receiver responds with DLE EOT, then take ?SSND error return (error code ERDIS) and begin BSC error-recovery procedures. - If receiver responds with other than NAK or DLE EOT, take ?SSND error return on error ERUNI and begin BSC error-recovery procedures. - If NAK within time-out interval, take ?SSND normal return.
Multipoint control station	Send initial (with data) (?CINT)	Send select sequence (pointed to by AC0) to receiver and wait for ACK0 response. - If bid successful, ?CINT is changed to ?CONT; otherwise, remains the same. - If no ACK0 within time-out interval, then begin BSC error-recovery procedures. - If ACK0 (acknowledging select sequence), then send data block and wait for ACK1. - If no ACK1 within time-out interval, then begin BSC error-recovery procedures. - If ACK1, then take ?SSND normal return.

(continued)

Table 2–199. ?SSND Call Types

Station Type	Call Type	Action
	Send continue (with data) (?CONT)	Functionally identical to point-to-point send continue calls.
Multipoint	Send initial (with data)	Not applicable. (You will never issue an ?SSND initial if you are a multipoint tributary; instead, you issue an ?SRCV initial to monitor the line for your poll address. If you are polled and have data to send, issue an ?SSND send continue and send the first block of data.)
	Send continue (with data) (?CONT)	Functionally identical to point-to-point send continue calls.

(concluded)

Notes

- See the description of ?SRCV in this chapter.

?STMAP
error return
normal return

Input

AC0 Contains the following:

- Bit 1 is a flag bit:

  Bit 1 = 0 if device is not using a map definition table

  Bit 1 = 1 if device is using a map definition table (defined with ?IDEF)

- Bits 0 and 2 through 31 contain the device code

AC1 Contains the following:

- Bit 0 = 0
- Bits 1 through 31 contain the address of the user's device buffer

AC2 If Bit 1 of AC0 is 1 (map definition table is used), flag bits:
High–order portion of buffer address
Bits 16–19 = map definition table in AC1 (16–bit processes only) entry number in range 0 through 7

Bits 20–25 = offset of first map slot to load in this map table entry

Bits 26–31 number of map slots to load

If Bit 1 of AC0 is 0 (map definition table is not used), AC2 is not used

Output

AC0 Unchanged

AC1 Contains the following:

- Bit 0 = 0
- Bits 1 through 31 contain the buffer's logical address

AC2 Unchanged (32–bit processes only)

If your program executes as a 16–bit process, change the following bit numbers in AC0, AC1, and AC2.

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

?STMAP Continued

Error Codes in AC0

ERDNM	Illegal device code
ERPRE	System call parameter address error
ERPRV	Caller not privileged for this action
ERPTY	Illegal process type (for example, the calling process is not resident)

Why Use It?

?STMAP works in conjunction with ?IDEF. That is, ?IDEF defines a user device, its I/O channel (BMC or DCH), and optionally, its map slots. ?STMAP initializes the map slots.

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

?STMAP initializes slots in the data channel (DCH) map or burst multiplexor channel (BMC) map for the device that you specify in AC0. ?STMAP also returns the first logical address of the device buffer.

Before you issue ?STMAP, perform the following steps:

1. Set up an I/O buffer for the data transfers in your logical address space. (Note that you must wire the buffer to your working set before you issue ?STMAP. See the description of ?WIRE in this chapter.)

2. Load the buffer address into Bits 1 through 31 of AC1 (Bit 0 must contain 0).

3. Load Bits 2 through 31 of AC0 with the device code you assigned in ?IDEF.

If you did not define a map definition table, all slots that you defined when you issued ?IDEF will be loaded when you issue ?STMAP. The operating system assumes contiguous user logical buffer addresses if you defined more than one slot at ?IDEF time.

If you defined a map definition table, set Bit 1 of AC0 to 1, and set AC2 by performing the following steps:

1. Load Bits 16 through 19 with the number of the map table entry for this device (the entries are numbered 0 through 7).

2. Load Bits 20 through 25 with the offset of the first map slot you want to initialize within the group that this entry allocates.

3. Load Bits 26 through 31 with the total number of map slots (range: 1 through 40 for DCH map; 1 through 63 for BMC map).

Bits 16 through 19 and 26 through 31 of AC2 correspond to the entry number and total number of map slots you defined in the map definition table (with ?IDEF).

Bits 20 through 25 of AC2 must contain an offset from the device's first map slot. The first map slot is the number the operating system returned to the map definition table entry (First Acceptable Slot field) as output to ?IDEF. Thus, if you want to initialize all the device's map slots, set Bits 20 through 25 to 0. If you want to initialize all slots except the first slot, set Bits 20 through 25 to 1.

Notes

- See the description of ?IDEF in this chapter.

?STOD
error return
normal return

Input

AC0	Seconds from 0 through 59
AC1	Minutes from 0 through 59
AC2	Hour from 0 through 23

Output

AC0	Unchanged
AC1	Unchanged
AC2	Unchanged

Error Codes in AC0

ERPRV	Caller not privileged for this action (The calling process is not PID 2.)
ERTIM	Illegal time argument (The second, minute, or hour value is outside the legal range.)

Why Use It?

?STOD, which is similar to ?SDAY, allows you (the operator at PID 2) to adjust the system clock at runtime.

Who Can Use It?

Only the operator process (PID 2), or a process that has System Manager privilege, can issue ?STOD. There are no restrictions concerning file access.

What It Does

?STOD sets the system clock to the second, minute, and hour values that you specify in AC0, AC1, and AC2, respectively.

?STOM
error return
normal return

Operating System Differences

See the section "What It Does" for the differences.

Input

AC0 One of the following:

- Channel number for the device
- Byte pointer to the name of the device

AC1 Flag bit:

- Bit 0 = 0 if AC0 contains a byte pointer
- Bit 0 = 1 if AC0 contains a channel number

AC2 One of the following:

- Time–out value in seconds
- –1 for the default time–out value

Output

AC0 Unchanged

AC1 Unchanged

AC2 Unchanged

Error Codes in AC0

ERICN	Illegal channel
ERIFT	Illegal file type IPC error codes
ERVBP	Invalid byte pointer passed as a system call argument

Why Use It?

You can use ?STOM to set a time–out value. This is the time that the operating system will wait for a response from the target device before it assumes that there is a problem with the device.

Who Can Use It?

There are no special process privileges needed to issue this call. You must be the current owner of any terminal that you specify.

What It Does

?STOM sets a time–out value (in seconds) for the device that you specify in AC0. A time–out value is the length of time the operating system will wait for a response from the target device before it takes an error return or begins error–recovery procedures. If the target device is a character one, the timeout–value applies to individual characters in a response — not to all the characters in a response. The shortest possible time–out value is 2 seconds. If you set the time–out to a value less than 2 seconds, the operating system sets it to 2 seconds. If the device allows input and output, the time–out value that you specify applies to both.

If the target device is a character device, the time–out value will have no effect until you select the time–out enable characteristic (?CTO) in the device characteristics word to enable time–outs. AOS/VS supports only devices that its PMGR software controls; AOS/RT32 supports only terminals.

Setting a time–out value lets you detect when a device is inactive. If you set a time–out value and issue a ?READ system call, the call waits until an input (or output completion for a plotter) interrupt is received or until the time–out period expires. If the time–out period expires before the interrupt is received the outstanding I/O call will take an error return and place ERDTO in AC0. For console devices the timeout applies to each character received until ?READ is satisfied.

If you set AC2 to –1, the operating system uses one of the following default time–out values:

Device	AOS/VS Default Time–out Value	AOS/RT32 Default Time–out Value
All terminals	(The OS does not check time–out.)	32 seconds
CDR (card reader)	2 seconds	Not Applicable
PLT (plotter)	2 seconds	Not Applicable

?SUPROC Enters, leaves, or examines Superprocess mode.

?SUPROC
error return
normal return

Operating System Differences

See the section "Who Can Use It" below.

Input

AC0	One of the following: • –1 to turn Superprocess mode on • +1 to turn Superprocess mode off • 0 to return the current status of Superprocess mode
AC1	Reserved (Set to 0.)
AC2	Reserved (Set to 0.)

Output

AC0	One of the following: • –1 if Superprocess mode is on • +1 if Superprocess mode is off
AC1	Undefined
AC2	Undefined

Error Codes in AC0

ERPRE	Invalid system call parameter
ERPRV	Caller not privileged for this action (The caller lacks privilege ?PVSP.)

Why Use It?

You can use ?SUPROC to enter or leave Superprocess mode. Superprocess mode permits a process to override the operating system's process protections. For example, a process might enter Superprocess mode to terminate (?TERM) or block (?BLKPR) an unrelated process or one of its superiors, such as its father.

Who Can Use It?

Under AOS/VS, your process must have privilege ?PVSP unless you only want to find out if your process is currently in Superprocess mode (i.e., examine Superprocess mode). For this use of ?SUPROC, no privilege is necessary. There are no restrictions concerning file access.

Note that an AOS/VS process in Superprocess mode can terminate any other process. Therefore, you should restrict privilege ?PVSP.

Under AOS/RT32, all processes have all privileges, and thus can issue any version of ?SUPROC.

What It Does

?SUPROC allows a calling process that has the ?PVSP privilege to enter or leave Superprocess mode. A process that is in Superprocess mode can issue ?BLKPR, ?BRKFL, ?CTYPE, ?PRIPR, ?TERM, or ?UBLPR system calls against any process, not just its subordinate processes.

A process remains in Superprocess mode until it terminates or until it issues another ?SUPROC to turn off Superprocess mode.

?SUS — Suspends the calling task.

?SUS
error return
normal return

Input

None

Output

None

Error Codes in AC0

No error codes are currently defined.

Why Use It?

You can use ?SUS to suspend the task that is currently executing and transfer control to another task — either the next ready task in the same priority level or the first ready task in the next 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

?SUS suspends the calling task until another task explicitly readies it with ?IDRDY or ?PRRDY, terminates it with ?IDKIL or ?PRKIL, or redirects it with ?IDGOTO.

Notes

- See the descriptions of ?IDRDY, ?PRRDY, ?IDKIL, ?PRKIL, and ?IDGOTO in this chapter.

?SUSER
error return
normal return

Operating System Differences

See the section "Who Can Use It" below.

Input

AC0 One of the following:

- –1 to turn Superuser mode on
- +1 to turn Superuser mode off
- 0 to return the current status of Superuser mode

AC1 Reserved (Set to 0.)

AC2 Reserved (Set to 0.)

Output

AC0 One of the following:

- –1 if Superuser mode is on
- +1 if Superuser mode is off

AC1 Undefined

AC2 Undefined

Error Codes in AC0

ERPRE	Invalid system call parameter
ERPRV	Caller not privileged for this action
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 ?SUSER as you would use ?SUPROC, except that it allows a process unrestricted access to files, rather than to processes.

Who Can Use It?

Under AOS/VS, your process must have privilege ?PVSU unless you only want to find out if your process is currently in Superuser mode (i.e., examine Superuser mode). For this use of ?SUSER, no privilege is necessary. There are no restrictions concerning file access.

Under AOS/VS, note that a process in Superuser mode has access to any file. Therefore, you should restrict privilege ?PVSU.

Under AOS/RT32, all processes have all privileges, and thus can issue any version of ?SUSER.

What It Does

?SUSER allows a process that has the ?PVSU privilege to enter or leave Superuser mode. Superuser mode gives a process the right to access any file on your system, regardless of the file's access controls.

?SUSER Continued

Once a process is in Superuser mode, it remains in that state until it terminates or until it issues another ?SUSER to turn off Superuser mode.

If you disable access controls (in the system–generation procedure), all processes run in Superuser mode. If you then attempt to turn off Superuser mode, the operating system takes the normal return with –1 in AC0.

Notes

- See the description of ?SUPROC in this chapter.

AOS/VS and AOS/VS II

?SYLOG
error return
normal return

Input

AC0 One of the following:

- ?SLRE (–2) to return the extended status of logging
- ?SLRS (–1) to return the current state of SYSLOG logging function
- ?SLSP (0) to stop logging system events in :SYSLOG
- ?SLST (1) to start logging system events in :SYSLOG
- ?SLDS (2) to stop soft error reporting (tape only)
- ?SLES (3) to start soft error reporting (tape only)
- ?SLTE (4) to transfer the contents of :ERROR_LOG to the new file named by the byte pointer in AC1.
- ?SLRF (5) to retrieve an active exclusion bit map.
- ?SLSF (6) to set an exclusion or superuser bit map.
- ?SLBC (7) to start logging CON0 I/O events in :CON0_LOG
- ?SLEC (8) to stop logging CON0 I/O events in :CON0_LOG
- ?SLCSU (9) to start or stop Superuser logging to :SYSLOG.

Output

AC0 One of the following:

- 0 if the system logging function is off
- 1 if the system logging function is on

If AC0 = ?SLRE on input, then on output AC0 will contain the following status bits:

Mask	Bit	Feature (0 =off, 1=on)
?SLCON	26	CON0 logging
?SLSEX	27	Superuser bit map
?SLSU	28	Superuser logging
?SLEX	29	exclusion bit map
?SLFL	30	minimal logging =0 full logging =1
?SLON	31	system logging

Bits 0–25 are reserved (Set to 0.)

?SYLOG Continued

AC1	One of the following: • Byte pointer to a name for the system log file (AC0 contains 1 or –1) • 0 to retain the current name • Byte pointer to the name of the file that receives the contents of the system error log file (AC0 contains 4) • Byte pointer to a name for the CON0 log file (AC0 contains 7)	AC1	Unchanged
AC2	If AC0 does not contain ?SLST, ?SLRF, ?SLSF, or ?SLCSU set to 0. If AC0 contains ?SLST, • Set to 0 to specify minimal (default) logging. • Set to –1 to specify full logging. If AC0 contains ?SLRF, set to the address of an exclusion bit map packet. The size of the buffer must be at least ?SYLEN words. If AC0 contains ?SLSF, • Set to the address of the exclusion bit map packet. • Set to –1 to delete any existing exclusion bit map. If AC0 contains ?SLCSU • Set to 0 to turn Superuser logging off. • Set to 1 to turn Superuser logging on.	AC2	Undefined

Error Codes in AC0

ERPRE	Invalid system call parameter.
ERPRV	Caller not privileged for this action. (The caller is not PID 2 or does not have SYSMGR privilege.)
ERVBP	Invalid byte pointer passed as a system call argument.
ERICM	Exclusion bit map not found, or log function is off while attempting to initialize map or to start Superuser logging.

Why Use It?

Use ?SYLOG to control and monitor the status of ongoing system, Superuser, error, and CON0 (AOS/VS II only) logging. You can use CON0 logging to write to a file, instead of to a hardcopy console. You can also use ?SYLOG to rename the system, the error, and the CON0 log files each day, helping you keep a continuous record of system and CON0 I/O activity.

Who Can Use It?

Only the operator process (PID 2), or a process with System Manager privilege, can issue ?SYLOG. Restrictions concerning file access control are described in the next section.

What It Does

The operator process uses ?SYLOG to find out the current state of each logging function (on or off), and to start and stop system, Superuser, and CON0 logging; you cannot stop error logging with ?SYLOG. System and Superuser logging are written to :SYSLOG including all messages sent by the operating system and/or EXEC, and all messages sent by users via ?LOGEV. Error logging is written to :ERROR_LOG, and CON0 logging to :CON0_LOG. All message logging is buffered in the operating system before it is written to a function's log file.

Stopping and restarting system, Superuser, and CON0 logging flushes each function's message buffer, writing the buffer's content into the function's log file. When the operator process restarts logging, messages are appended to each function's log file. The operator process can use ?SYLOG to create or rename :SYSLOG and :CON0_LOG, but can only rename :ERROR_LOG.

Renaming a log file also flushes its message buffer, and copies the old log file's content plus the buffer's content into the newly named file. You can use renaming to create a continuous record of system activity. Renaming :ERROR_LOG reduces its size to zero bytes, and writes the file plus the buffer content into the newly named file.

When renaming the log file while system logging is enabled, first issue an extended status call to obtain the current level of logging detail. Then issue the rename call with the appropriate detail level set for logging in AC2. When the log file is renamed during a status command, and AC0=–1, AC2 must contain the desired level of logging detail (0 or –1) as well. If you omit the logging detail level, the operating system sets a minimal detail level after renaming the log file, and you could inadvertently change your logging detail level.

You can view :SYSLOG and :ERROR_LOG with the REPORT utility. You can view the ASCII file :CON0_LOG using the CLI TYPE command, or the BROWSE utility. You view a function's most recent messages by renaming the log file with ?SYLOG. For example, to view all the error messages up to the current time, rename :ERROR_LOG. Renaming the file flushes the error log buffer, and writes the most recent error messages and the system time and date stamp into the newly named file. Now use the REPORT utility to view the renamed file's content.

For information about reading :SYSLOG with a program, see Appendix B at the end of *AOS/VS, AOS/VS II, and AOS/RT32 System Call Dictionary, ?R Through ?Z* (093–000543–02).

The AC0 and AC1 input values determine which function ?SYLOG performs. If you use ?SYLOG to start the system logging function (AC0 = 1) and the system log file does not exist, the operating system creates :SYSLOG. If you use ?SYLOG to start the CON0 logging function (AC0 = 7) and the CON0 log file does not exist, the operating system creates :CON0_LOG.

The access control list for :SYSLOG is NULL, giving only processes with Superuser privilege access to the file and denying access to other users. The :SYSLOG special file type is ?FLOG.

?SYLOG Continued

The access control list for :CON0_LOG is OP,R giving the operator process read access to the file, and denying access to other users. The :CON0_LOG special file type is also ?FLOG. With CON0 logging on, the operating system buffers and then writes all CON0 I/O messages to the file :CON0_LOG.

If you turn off system logging (AC0 = 0), you flush the operating system buffer and close :SYSLOG. Turning system logging off prevents the operating system from logging any further system events or messages to :SYSLOG. Similarly, if you turn off CON0 logging (AC0 = 0), you flush the operating system buffer and close :CON0_LOG. Turning CON0 logging off prevents the operating system from logging any further I/O events or messages to :CON0_LOG.

To rename :SYSLOG, load AC1 with a byte pointer to the new filename before you issue ?SYLOG. The operating system closes the current :SYSLOG file, if it is open, and then renames the file.

If you leave the system logging function on while you rename :SYSLOG, or if you rename the log file and then issue another ?SYLOG to resume system logging, the operating system opens a new system log file called :SYSLOG and writes all subsequent log messages to that file. All previous log files are strictly historical.

You cannot rename :CON0_LOG while CON0 logging is active. To rename :CON0_LOG, turn off CON0 logging (AC0=0), turn on Superuser and load AC1 with a byte pointer to the new filename before you issue ?SYLOG. Now turn off Superuser and restart CON0 logging to :CON0_LOG.

If an error occurs during ?SYLOG, the operating system leaves the current state of the system log file and its name intact.

Exclusion Bit Map

An exclusion packet can be used with ?SYLOG to selectively ignore certain events. Figure 2–243 shows the structure of the exclusion bit map packet. The exclusion packet is a bit map record. The ordinal bit positions are defined in PARU.32.SR from ?LSMI to ?LUMAX (Only ?LSMI to ?LSMA apply to Superuser logging exclusion bit maps.).

To retrieve and view the current bit map settings before changing them, use ?SLRF in AC0. In the first double word (?SYFBM) of the exclusion bit map packet define either ?SYLID or ?SYSUID. ?SYLID indicates a a selective bit map, ?SYSUID indicates a Superuser bit map. When setting an exclusion or a Superuser bit map with ?SLSF, if a bit is set in the packet, the system will not log that event.

You can set exclusion bit maps only while system logging is on, and you can set Superuser exclusion bit maps only while Superuser logging is on. Also, Superuser exclusion maps do not affect PMGR system logging events.

The operating system writes the "Set Exclusion Map" message in a SYSLOG entry with code ?LTSF.

NOTE: We recommend that you always include the AOS/VS administration and hardware codes (1–899.).

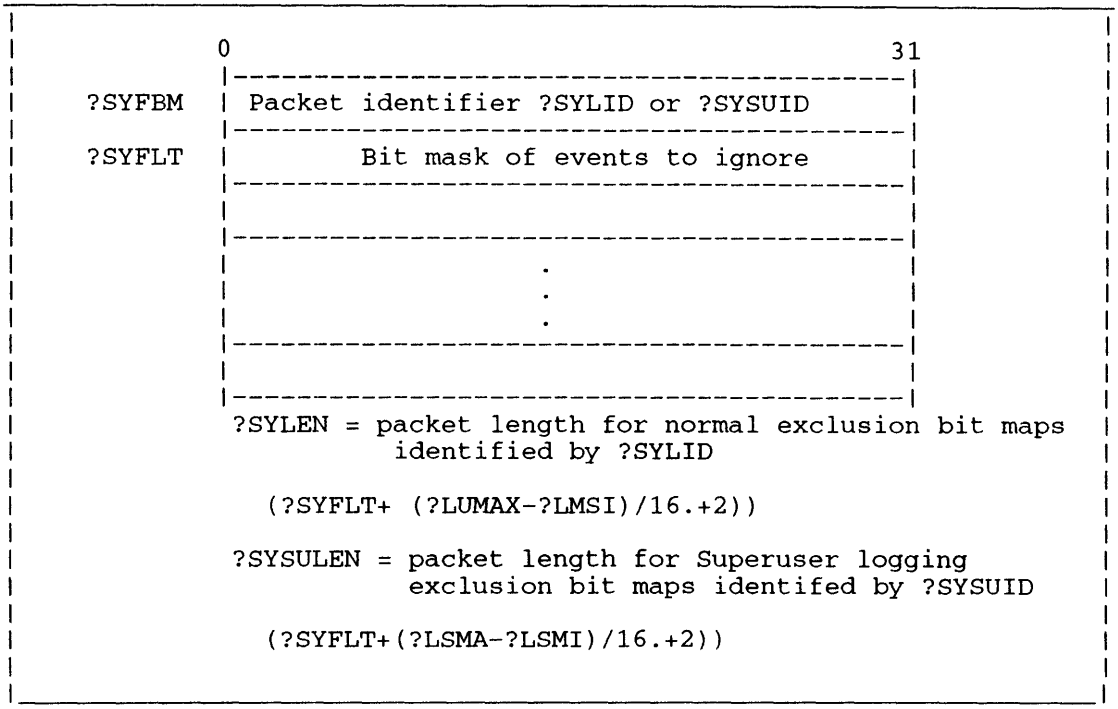


Figure 2–226. Structure of ?SYLOG Exclusion Bit Map Packet

The first two words in the packet contain the system call identifier ?SYLID or ?SYSUID. Place ?SYLID here for ordinary exclusion bit maps, or place ?SYSUID here for bit maps used for superusers during Superuser logging. The remainder of the packet is a bit map record that contains a bit entry for every possible log event. Bit positions correspond to the event codes that are defined in PARU.32.SR. To ignore specific events, set the corresponding bit for that event in the packet.

Notes

- See the description of ?LOGEV in this chapter.
- See the explanation of the CLI command SYSLOG in the manual *Managing AOS/VS and AOS/VS II*.
- Stopping ?SYLOG events or using an exclusion bit map precludes a C2–level secure system.

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

Operating System Differences

See the section "Who Can Use It" below.

Input

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

Output

AC0	Unchanged
AC1	Unchanged
AC2	Address of the ?SYSPRV packet

Error Codes in AC0

ERPRE	Invalid system call parameter
ERPRV	Caller not privileged for this action
ERPVM	Unknown privilege mode
ERPVO	Privilege cannot be held exclusively
ERPVP	Other processes using privilege
ERPVX	Privilege held exclusively by other process

Why Use It?

Use this system call to inquire about or change certain process privilege states. You need certain privileges before you can execute some system calls. For example, your process must have System Manager privilege turned on before it can issue a ?JPINIT call to initialize a job processor.

Who Can Use It?

Under AOS/VS, there are no restrictions concerning file access. The following list contains the AOS/VS process privileges needed to issue ?SYSPRV.

- To enter (i.e., acquire) a privilege state — if you are allowed to have one of the privileges Superuser, Superprocess, or System Manager and issue ?SYSPRV to enter the respective privilege, AOS/VS lets you do this. (One of the ways you are allowed to have, say, Superuser privilege is for your system manager to use the profile editor utility program, PREDITOR, to allow you this privilege.) If you are not allowed to have one of these privileges and issue ?SYSPRV to enter the privilege, AOS/VS returns error ERPRV.

- To enter (i.e., acquire) the System Manager privilege state exclusively — you must be allowed to have System Manager privilege and no other user has entered System Manager privilege.

- To leave a privilege state — if you already have entered one of the privilege states Superuser, Superprocess, or System Manager and issue ?SYSPRV to leave the respective state, AOS/VS allows you to do this. If you have not entered one of these privilege states and issue ?SYSPRV to leave the privilege, AOS/VS takes no action.
- To examine a privilege state — you need no special privileges.

Under AOS/RT32, all processes have all privileges, and thus can issue any version of ?SUPROC.

What It Does

This system call reports on or modifies your process's privileges. These privileges — each of which is on or off — are Superuser, Superprocess, and System Manager. If you want to change more than one privilege, then issue ?SYSPRV repeatedly, with one change each time, until you've made all the changes.

Figure 2–227 shows the structure of the ?SYSPRV parameter packet, and Table 2–200 describes its contents.

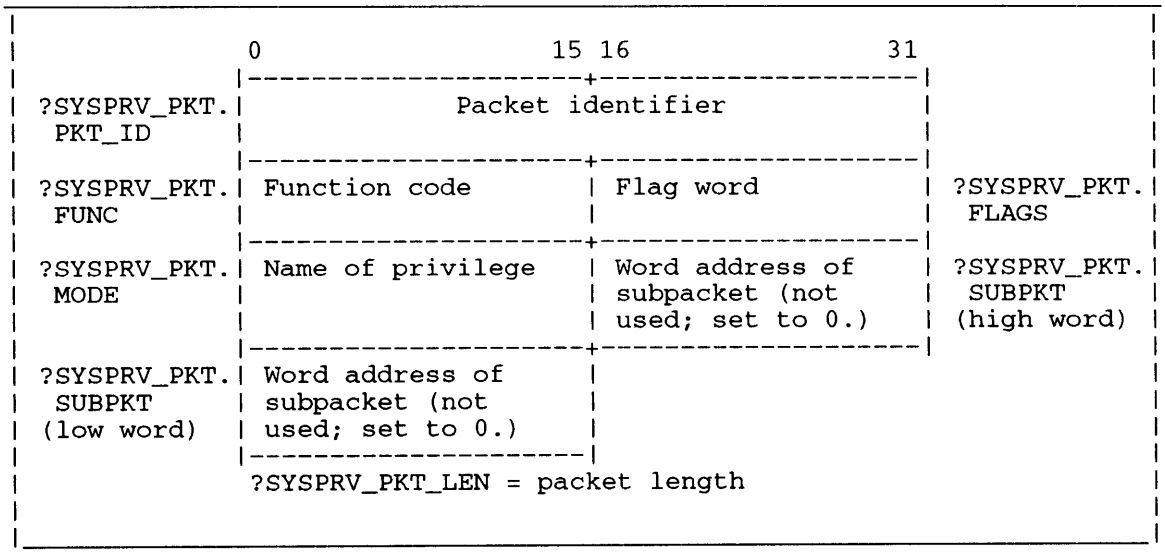


Figure 2–227. Structure of ?SYSPRV Packet

Table 2–200. Contents of ?SYSPRV Packet

Offset	Contents
?SYSPRV_PKT.PKT_ID (doubleword)	Packet identifier. Place ?SYSPRV_PKT_PKTID here.
?SYSPRV_PKT.FUNC	Function code. The following values have the effects next to them. ?SYSPRV_GET — Get privilege status. ?SYSPRV_ENTER — Enter a privileged mode. ?SYSPRV_ENTER_EXCL — Enter a privileged mode exclusively. ?SYSPRV_LEAVE — Leave a privileged mode.
?SYSPRV_PKT.FLAGS	Flag word. AOS/VS uses it only if offset ?SYSPRV_PKT.FUNC contains ?SYS_PRV_GET. In this case, AOS/VS returns the truth values (1=true, 0=false) of the following bits. ?SYSPRV.FLAGS.CALLER — Caller has privilege on. ?SYSPRV.FLAGS.CALLER_EXCL — Caller has privilege on exclusively. ?SYSPRV.FLAGS.OTHERS — Other processes have privilege on. ?SYSPRV.FLAGS.OTHERS_EXCL — Other processes have the privilege on exclusively.
?SYSPRV_PKT.MODE	Privilege code. ?SYSPRV_SUSER — Superuser mode. ?SYSPRV_SPROCESS — Superprocess mode. ?SYSPRV_SYSMGR — System manager mode.
?SYSPRV_PKT.SUBPKT (doubleword)	Word address of subpacket. Currently, there is no subpacket, so set this doubleword to zero.

Notes

- Next is the correspondence between ?SYSPRV and equivalent system calls:
  - ?SUSER is the equivalent of ?SYSPRV with the Superuser privilege.
  - ?SUPROC is the equivalent of ?SYSPRV with the Superprocess privilege.
  - ?PSTAT is the equivalent of ?SYSPRV with the value ?SYSPRV_GET in offset ?SYSPRV_PKT.FUNC.
- A process remains in a privileged state until it terminates or until it turns that state off with another ?SYSPRV call.
- The system log records each execution of ?SYSPRV.

?TASK *[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 (TDP), unless you specify the address as an argument to ?TASK

Output

AC0	Undefined
AC1	Undefined
AC2	Address of the task definition packet (TDP)

Error Codes in AC0

ERNOT	No task control block available
ERSTS	Stack too small
ERTID	Task ID error
ERTMT	Too many tasks requested
ERVWP	Invalid word pointer passed as a system call argument

Why Use It?

?TASK allows you to create new tasks. By creating multiple tasks, you can take full advantage of the operating system's multitasking environment.

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

?TASK initiates one or more tasks to create a multitasking environment. If the ?TASK caller specifies a starting PC within Ring 7, then control passes to a task initiation routine called ?UTSK. This occurs during the initiation sequence invoked by ?TASK.

By default, the operating system uses a dummy ?UTSK routine in the user runtime library URT32.LB. This routine returns control immediately to the task scheduler, which initiates the tasks. As an alternative, you can supply your own ?UTSK task initiation routine and link it with your program. (See "Task Initiation" for more information on task initiation routines.)

When the operating system starts executing a new task, it loads AC3 with a pointer to the system task termination code. You should begin the code for each task (except the initial task) with a WSSVS or a WSSVR instruction, and end the code with a WRTN instruction. Do not use the WSSVS/WSSVR and WRTN sequence for the initial task.

Each ?TASK requires a task definition packet, which you must define in your logical address space. There are two packet formats: a standard format and an extended format for queued task creation. The symbol ?DSLTH represents the standard packet's length. Figure 2–228 shows the structure of the standard task definition packet and Table 2–201 describes the contents of each offset.

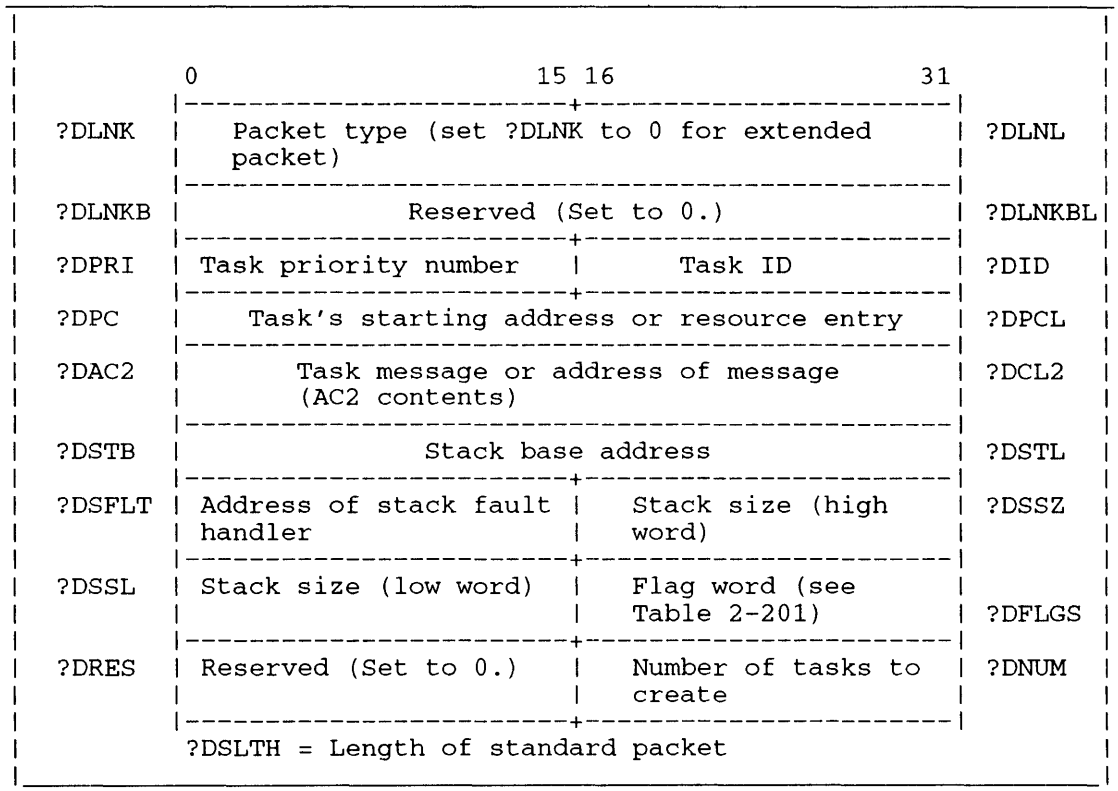


Figure 2–228. Structure of Standard Task Definition Packet

Table 2–201. Contents of Standard Task Definition Packet

Offset	Contents
?DLNK (doubleword)	Packet type. Set to a nonzero value for the standard packet.
?DLNKB (doubleword)	Reserved. (Set to 0.)
?DPRI	Task priority number. Set to 0 for same priority as calling task.
?DID	Task ID # (0, 1-32). Set to 0 for no Task ID.
?DPC (doubleword)	Starting address of task or resource entry.
?DAC2 (doubleword)	Task message or address of message (initial AC2 contents.)
?DSTB (doubleword)	Stack base address. Set to -1 for no stack.
?DSFLT	Address of stack fault handler. Set to -1 for the current address.
?DSSZ	Stack size (high word). The OS ignores this offset if there is no stack.
?DSSL	Stack size (low word).
?DFLGS	Task flag word. ?DFLRC--Resource call task.
?DRES	Reserved. (Set to 0.)
?DNUM	Number of tasks you want to create.

Offset ?DLNK defines the packet type. If you are using the extended packet, set the one–word field ?DLNK to 0. If you select the standard packet, ?DLNK can contain any nonzero value.

Set offset ?DNUM to the number of tasks you are creating with ?TASK. You can define as many as 32 potential tasks per program, either at assembly time or at link time.

Offset ?DPRI contains the priority number of the newly created task or tasks. If you default this parameter by setting it to 0, the operating system assigns the new task(s) the same priority as the calling task. The packet contains only one parameter for task priority. Therefore, if you use a single ?TASK to initiate more than one task, the operating system initiates all of the new tasks at the same priority level.

Offset ?DID contains the TID of the new task. Do not set ?DID to a TID that is already in use.

If you are using ?TASK to initiate more than one task, the value of ?DID becomes the TID for the first task and the base for the other TIDs. That is, the operating system increments the ?DID value by 1 for each additional TID. When you default this parameter (to 0), the operating system does not assign TIDs to the new tasks.

Offset ?DPC contains the starting address of the task code; that is, the location where execution will begin. This can be an address within your unshared logical context or an address within a shared routine.

You can use the task definition packet to send a 32–bit message, via AC2, to each new task as soon as it assumes control. If you are initiating only one task with ?TASK, place the message in offset ?DAC2. If you are initiating more than one task, offset ?DAC2 should point to a message table that contains one 32–bit message for each task. If you do not want to send task messages, set offset ?DAC2 to 0.

Offset ?DSTB specifies the stack base; that is, the starting address of the new task's stack. Offset ?DSSZ indicates the stack size, which must be at least 60 words for a 32–bit program. If you set ?DSTB to –1 (the default), which specifies no stack, you must allocate the stack after the task begins executing.

Only the Ring 7 stack base addresses can be supplied to ?TASK. This is true even if ?TASK is issued from a ring other than Ring 7. (See the description of the ?RINGLD system call in this chapter for more information on stack support in rings other than Ring 7.)

If you are initiating more than one task, you must define a stack at least as large as ?DNUM*?DSSZ; that is, the total number of tasks (?DNUM) times the stack size. The operating system builds the stacks contiguously, offset from the same stack base. Figure 2–229 compares the stacks generated for single–task ?TASK system calls and multitask ?TASK system calls.

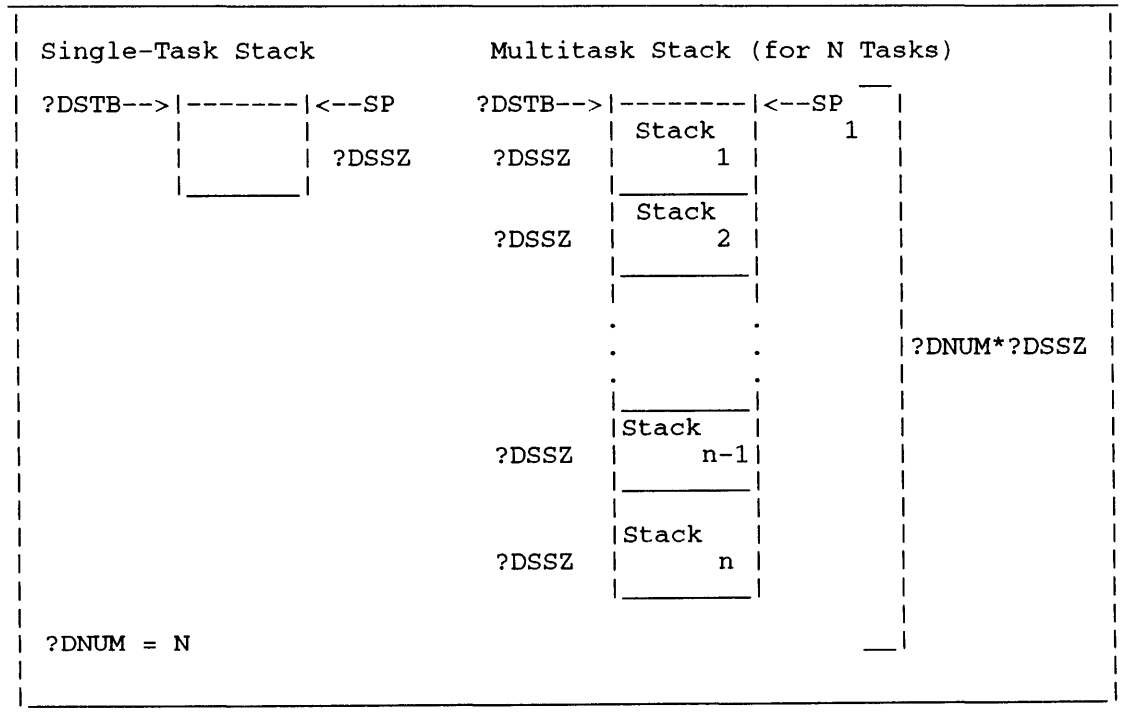


Figure 2–229. Stack Parameters for Initiating One or More Tasks

Offset ?DSFLT points to the address of the stack fault handler. The stack fault handler is a routine that gains control when there is a stack error. You can define your own stack fault handler or you can use the default stack fault handler in URT32.LB. To use the default stack fault handler, set ?DSFLT to –1. ?DSFLT is a 16–bit value. Therefore, the stack fault handler must be in the first 64K words of the ring.

Extended Task Definition Packet

The ?TASK extended packet contains all the parameters specified in the standard task packet, plus additional information for the queued task creation option. The extended packet is ?DXLTH words long. Figure 2–230 shows the structure of the extended packet, and Table 2–202 describes the contents of each offset.

Before you initiate queued tasks, issue the ?IQTSK system call, which creates a queued task manager to handle the queue. The ?DQTSK system call removes one or more task definition packets from a task creation queue.

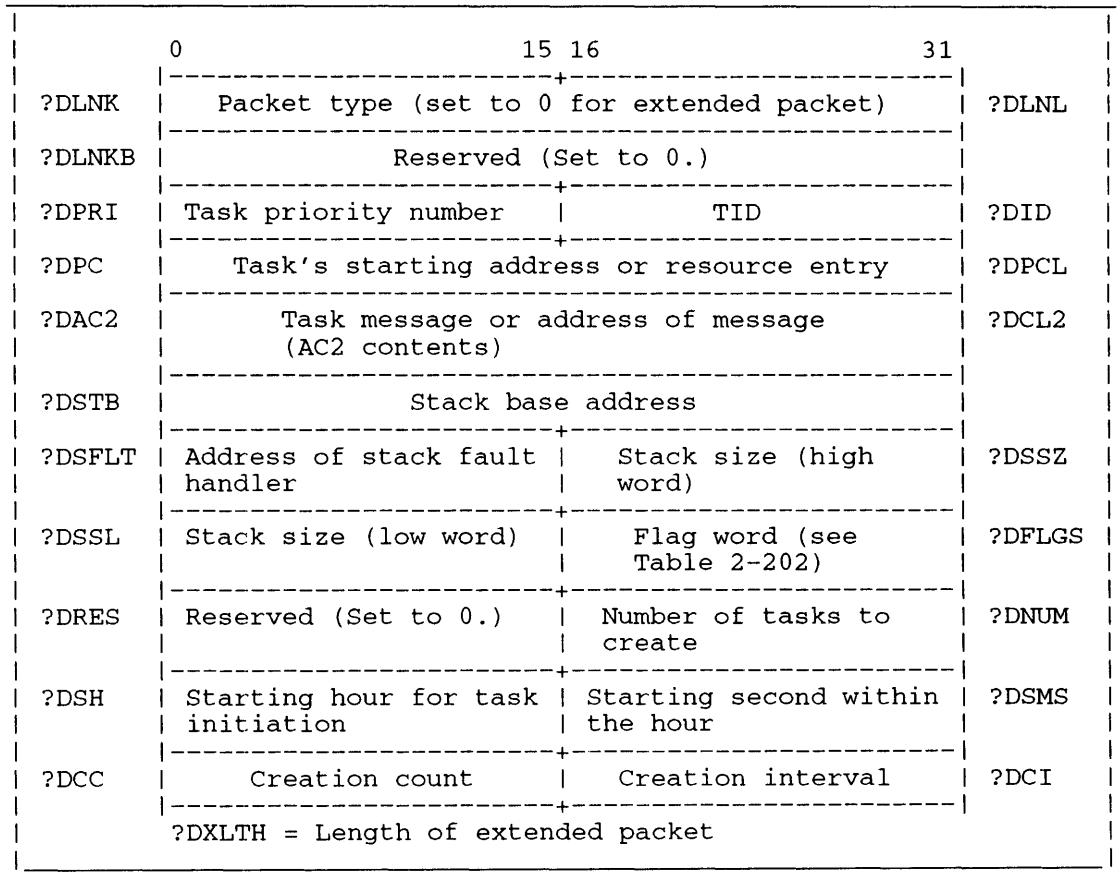


Figure 2–230. Extended Task Definition Packet

Offset ?DSH, starting hour, specifies the hour at which the operating system will begin initiating the new tasks. If you set this parameter to –1, the default, the operating system initiates the new tasks immediately and ignores the ?DSMS parameter.

Offset ?DSMS tells the operating system which second within the hour to start initiating the tasks. The range for this parameter is 1 through 3600 (there is no default value).

Table 2–202. Contents of Extended Task Definition Packet*

Offset	Contents
?DLNK (doubleword)	Packet type. Set to 0 for the extended packet.
?DLNKB (doubleword)	Reserved. (Set to 0.)
?DPRI	Task priority number. Set to 0 for same priority as calling task.
?DID	Task ID # (0, 1-32). Set to 0 for same priority as calling task.
?DPC (doubleword)	Starting address of task or resource entry.
?DAC2 (doubleword)	Task message or address of message (initial AC2 contents.)
?DSTB (doubleword)	Stack base address. Set to -1 for no stack.
?DSFLT	Address of stack fault handler. Set to -1 for the current address.
?DSSZ	Stack size (high word). The OS ignores this offset if there is no stack.
?DSSL	Stack size (low word).
?DFLGS	Task flag word. ?DFLRC--Resource call task.
?DRES	Reserved. (Set to 0.)
?DNUM	Number of tasks you want to create.
?DSH	Starting hour for task initiation. Set to -1 to initiate the task immediately.
?DSMS	Starting second within the hour for task initiation. (The OS ignores this offset if you set ?DSH to -1 for immediate task initiation.)
?DCC	Number of times to initiate the task(s).
?DCI	Amount of time between initiating tasks.

* There is no default unless otherwise specified.

Offset ?DCC specifies the number of times you want the system to repeat the task initiation(s). Set this parameter to –1 if you want the system to initiate the tasks repeatedly at the time intervals defined in ?DCI. Set ?DCC to 1 if you want to initiate the task(s) only once.

Offset ?DCI specifies the creation interval; that is, the number of seconds between each reinitiation. There is no default value for this parameter.

Sample Packet

The following sample packet creates three tasks.

```
PKT:     .BLK    ?DSLTH          ;Allocate enough space for this
                                 ;packet.  Packet length = ?DSLTH.
         .LOC    PKT+?DLNK       ;Specify extended or standard packet.
         .WORD   1               ;Standard packet (can contain any
                                 ;nonzero number).
         .LOC    PKT+?DLNKB      ;Reserved.
         .WORD   0               ;You must set this value to 0.
         .LOC    PKT+?DPRI       ;Specify priority of new tasks.
         .WORD   2               ;Medium priority. (Specify 0 for same
                                 ;priority as calling task.)
         .LOC    PKT+?DID        ;Specify TID of first task.
         .WORD   2               ;TID is 2 for first task. (The OS
                                 ;increments the TID of each
                                 ;additional task by 1.)
         .LOC    PKT+?DPC        ;Specify task's starting address.
         .DWORD  STRT            ;Starting address of each new task is
                                 ;STRT.  The new tasks execute the
                                 ;same code.
         .LOC    PKT+?DAC2       ;Specify any task messages.
         .DWORD  MESTAB          ;MESTAB is a table that contains a
                                 ;different message in AC2 for each
                                 ;task.
         .LOC    PKT+?DSTB       ;Stack base address.
         .DWORD  STKBAS          ;Stack base address is STKBAS.
         .LOC    PKT+?DSSZ       ;Size of stack for each task.
         .DWORD  STKSZ           ;Stack size is STKSZ.
         .LOC    PKT+?DSFLT      ;Address of stack fault handler.
         .WORD   -1              ;Use the OS default stack handler,
                                 ;URT32.LB.
         .LOC    PKT+?DFLGS      ;Task flag word.
         .WORD   0               ;There is no task flag word.
         .LOC    PKT+?DRES       ;Reserved.
         .WORD   0               ;You must set this value to 0.
         .LOC    PKT+?DNUM       ;Number of tasks to start?
         .WORD   3               ;Start 3 tasks.
         .LOC    PKT+?DSLTH      ;End of packet.
STKBAS:  .BLK    STKSZ*3         ;When you declare the stack base
                                 ;address for the first task, the OS
                                 ;divides it by the number of tasks
                                 ;you are starting.
```

Notes

- See the description of ?RINGLD in this chapter for information on stack support in rings other than Ring 7.
- See the descriptions ?IQTSK and ?DQTSK in this chapter.

?TERM
error return
normal return

Input

AC0 One of the following:

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

AC1 One of the following:

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

Otherwise, the contents of AC1 are ignored.

AC2 One of the following:

- Word address of the IPC message header
- 0 if there is no message header

Output

AC0 Unchanged

AC1 Unchanged

AC2 Unchanged

NOTE: AC2 input values are valid only if the calling process issues ?TERM to terminate itself.

Error Codes in AC0

ERMPR	System call address error
ERPNM	Illegal process name format
ERPRE	Invalid system call input, AC0 is not –1 and AC1 is not 0 or –1
ERPRH	Attempt to access a process not in the hierarchy
ERVBP	Invalid byte pointer passed as a system call argument

Why Use It?

?TERM allows you to explicitly terminate a process. Unless ?RETURN or ?TERM terminate a process, the process continues to exist until either the last task in the process kills itself, or the process traps because of a programming or system error.

Who Can Use It?

There are no special process privileges needed to issue this call to terminate the calling process or any subordinate process. If the calling process is in Superprocess mode, it can issue ?TERM to terminate any other process. There are no restrictions concerning file access.

What It Does

When a process terminates, so do all its offspring (sons, grandsons, and so forth). A process can terminate a subordinate even if the subordinate has a higher priority number or is a different process type.

If a process terminates itself, the operating system sends its father an IPC termination message. This IPC termination message is either a standard termination message or the message that you specify in AC2. In the latter case, the operating system ignores the ports designated in the IPC header.

Notes

- See the description of ?RETURN in this chapter.

?TIDSTAT Returns status of target task (32–bit processes only).

?TIDSTAT
error return
normal return

Input

AC0	Reserved (Set to 0.)
AC1	TID of task whose status you want to investigate
AC2	Reserved (Set to 0.)

Output

AC0	Status word
AC1	Unchanged
AC2	Undefined

Error Codes in AC0

ERTID	Task with specified task ID (TID) not found
ERICM	16–bit program incorrectly attempted to call ?TIDSTAT

Why Use It?

You can use ?TIDSTAT to find out what is happening to 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

?TIDSTAT returns a word in AC0 that describes the target task's status. (Refer to PARU.32 and PARU_LONG for a complete list of the currently defined status words. For example, ?TSSP is a bit setting to indicate that the target task is suspended.)

?TLOCK
error return
normal return

Input

AC0 ?TMYRING flag. If set, locks caller against task–redirection system calls issued from its own ring

AC1 Reserved (Set to 0.)

AC2 One of the following:

- Address of a double–word mailbox to contain notification of a pended redirection attempt
- –1 to indicate that the caller does not want to specify a mailbox

Output

AC0 Flag bits:

- ?TALOCK — Ring was already locked against task redirection when ?TLOCK was issued. If ?TALOCK is not set, there were no previous locks in effect for this ring.
- ?TMYRING — Ring was already locked against task–redirection system calls issued from caller's ring (This bit is only set if ?TALOCK is also set.)

AC1 Undefined

AC2 Address of the old mailbox (if any)

Error Codes in AC0

ERPRE	Invalid system call parameter
ERVWP	Invalid word pointer passed as a system call argument

Why Use It?

?TLOCK allows you to protect inner–ring segments from being redirected by ?IDGOTO, ?IDKIL, ?PRIKIL, ?IDSUS, and ?PRSUS task–redirection system calls issued from its own or higher rings.

Who Can Use It?

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

?TLOCK Continued

What It Does

?TLOCK accepts an input argument in AC2 that specifies the address of a double–word mailbox. If you provide a mailbox address, the operating system can inform a locked task that another task is trying to redirect it. Then, the operating system places a nonzero flag in the mailbox when a task–redirection request by another task pends.

A task that is executing within an inner–ring segment image can poll the mailbox, decide whether to abort its progress in the critical region, and then issue ?TUNLOCK to allow the redirection request to proceed.

?TLOCK can lock a task against being redirected by task–redirection system calls issued from its own ring. To do this, the caller must specify the ?TMYRING flag in AC0 on input.

The operating system can set the ?TMYRING flag in AC0 on output. In this case, the operating system sets ?TMYRING if the caller's ring was already locked against task–redirection system calls issued from within the same ring specified at ?TLOCK input. However, the operating system sets ?TMYRING on output only if ?TALOCK is also set.

Successive ?TLOCK system calls can change the state of task–redirection protection. When ?TLOCK changes the address of the mailbox associated with the calling task in the ring, the operating system places the current contents of the old mailbox in the new mailbox. Each successive ?TLOCK outputs the values of the state variables as they existed before the task issued ?TLOCK. Every task that uses ?TLOCK must save the values of these state variables until it leaves its critical region. Then, after it has left its critical region, each task must appropriately restore the state variables.

?TLOCK allows callers to retain task–redirection protection while changing either the mailbox address or the ?TMYRING variable. To do this, however, the caller's initial ?TLOCK must indicate that task–redirection protection was not already in effect when the task entered its critical region.

Consecutive ?TLOCKs from the same ring on behalf of a single task are legal, but they do not provide further protection. If you issue ?TLOCK from a ring 7 process, you must set Bit ?TMYRING.

To cancel the effect of a ?TLOCK, a task must issue ?TUNLOCK with the output flags from ?TLOCK as the input flags to ?TUNLOCK.

Notes

- See the description of the ?TUNLOCK system call in this chapter.

We removed the system call ?TMSG from the System Call Dictionary. Programs using ?TMSG work as they did before we removed ?TMSG.

Use ?IREC for old A–type termination messages, and for new B– and C–type termination messages.

We removed the system call ?TMSG from the System Call Dictionary. Programs using ?TMSG work as they did before we removed ?TMSG.

Use ?IREC for old A–type termination messages, and for new B– and C–type termination messages.

We removed the system call ?TMSG from the System Call Dictionary. Programs using ?TMSG work as they did before we removed ?TMSG.

Use ?IREC for old A–type termination messages, and for new B– and C–type termination messages.

We removed the system call ?TMSG from the System Call Dictionary. Programs using ?TMSG work as they did before we removed ?TMSG.

Use ?IREC for old A–type termination messages, and for new B– and C–type termination messages.

We removed the system call ?TMSG from the System Call Dictionary. Programs using ?TMSG work as they did before we removed ?TMSG.

Use ?IREC for old A–type termination messages, and for new B– and C–type termination messages.

We removed the system call ?TMSG from the System Call Dictionary. Programs using ?TMSG work as they did before we removed ?TMSG.

Use ?IREC for old A–type termination messages, and for new B– and C–type termination messages.

We removed the system call ?TMSG from the System Call Dictionary. Programs using ?TMSG work as they did before we removed ?TMSG.

Use ?IREC for old A–type termination messages, and for new B– and C–type termination messages.

We removed the system call ?TMSG from the System Call Dictionary. Programs using ?TMSG work as they did before we removed ?TMSG.

Use ?IREC for old A–type termination messages, and for new B– and C–type termination messages.

We removed the system call ?TMSG from the System Call Dictionary. Programs using ?TMSG work as they did before we removed ?TMSG.

Use ?IREC for old A–type termination messages, and for new B– and C–type termination messages.

We removed the system call ?TMSG from the System Call Dictionary. Programs using ?TMSG work as they did before we removed ?TMSG.

Use ?IREC for old A–type termination messages, and for new B– and C–type termination messages.

?TPID
error return
normal return

Input		Output	
AC0	PID or virtual PID	AC0	Local PID
AC1	Reserved (Set to 0.)	AC1	One of the following: • 0 if input PID is local PID • Host ID in Bits 17 through 31 if input PID is virtual PID • –1 if PID is remote and host ID is 0
AC2	Reserved (Set to 0.)	AC2	Unchanged

Error Codes in AC0

ERPRE Invalid system call parameter (You supplied an invalid PID in AC0.)

Why Use It?

?TPID decodes a PID into a local PID and a host ID. From this information, you can determine the process and hostname to which the virtual PID refers.

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

?TPID translates a virtual PID into its component parts: a local PID and a host ID. ?TPID also tells the caller when the input parameter is strictly a local PID. If ?TPID returns 0 in AC1, the input to AC0 was a local PID.

?TPORT Translates a local port number to its global equivalent.

?TPORT
error return
normal return

Input

AC0	Reserved (Set to 0.)
AC1	Reserved (Set to 0.)
AC2	Caller's local port number

Output

AC0	Undefined
AC1	32–bit global equivalent of the port number in AC2
AC2	Unchanged

Error Codes in AC0

ERIVP	Invalid port number

Why Use It?

A process can issue ?TPORT to determine the global port number assigned to one of its local ports. This information can be useful if you are sending and receiving over multiple ports or if you intend to issue an ?IS.R 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

?TPORT returns the global equivalent of the local port number you specify in AC2. The local port must belong to the ?TPORT caller.

Notes

- See the description of ?IS.R in this chapter.

?TRCON Reads a task message from the process terminal.

AOS/VS

?TRCON
error return
normal return

Input

AC0	Byte pointer to the message area (which must be word–aligned)
AC1	Reserved (Set to 0.)
AC2	Reserved (Set to 0.)

Output

AC0	Unchanged
AC1	Message byte count (including the New Line terminator, excluding the TID and commas)
AC2	Undefined

Error Codes in AC0

ERNOT	No task control block available
ERVBP	Invalid byte pointer passed as a system call argument

Why Use It?

?TRCON allows you to pass input from the process terminal to individual tasks in a multitasking 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

?TRCON prepares the calling task to receive a message that you type at your terminal. ?TRCON is incompatible with ?READ. Therefore, if you use ?TRCON to receive a message, you cannot issue a simultaneous ?READ against that terminal.

More than one task in a program can issue ?TRCON, because the operating system uses task identifiers (TIDs) to differentiate between the receivers. To use ?TRCON, a task must have a TID.

When you issue ?TRCON, the operating system initiates a message–manager task to monitor the process terminal for ?TRCON messages. The operating system terminates the message manager if it is still active after all active ready tasks in the process have terminated. (The message manager counts as one of the 32 possible tasks in the program.)

When the message manager is ready to receive a message, the operating system displays a > prompt on the process terminal. ?TRCON returns normally when the calling process receives the message.

Use the following format for the task messages:

<task–ID><comma>message<New Line>

where

task–ID	is the TID (in decimal) of the receiving task.
comma	is a required comma (,) as a delimiter.
message	is the message string.
New Line	is a required New Line terminator.

The message string can be up to 132 characters long, including the New Line terminator.

If there is an input error, the operating system returns one or more of the following messages to the process terminal:

- ***TID NOT FOUND*

  No task with the specified TID issued ?TRCON to receive the message.

- ***INPUT ERROR*

  You used an alphabetic character in the TID

- ***NO TASK WAITING*

  No task has issued ?TRCON

Notes

- See the description of ?READ in this chapter.

?TRUNCATE — Truncates a file at the current position.

?TRUNCATE
error return
normal return

Input

AC0	Reserved (Set to 0.)
AC1	Channel number
AC2	Reserved (Set to 0.)

Output

AC0	Undefined
AC1	Unchanged
AC2	Undefined

Error Codes in AC0

ERIFT Illegal file type
File system error codes

Why Use It?

?TRUNCATE allows you to control the size of a disk or magnetic tape file that is currently open for I/O. For disk files, for example, ?TRUNCATE allows the operating system to reclaim the disk storage that you no longer require.

Who Can Use It?

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

What It Does

?TRUNCATE truncates disk or magnetic tape files at the file pointer's current position. (The ?TRUNCATE caller must have previously issued ?OPEN to open the selected file.) There is no packet associated with ?TRUNCATE.

When you issue ?TRUNCATE against an open magnetic tape file, the operating system does not actually truncate the file. Instead, to indicate the end of the file, it writes two end–of–file marks (EOF) after the file pointer's position.

If the caller passes a channel number of an open channel that indicates the null file type (@NULL), ?TRUNCATE produces no action, but returns successfully. If the caller passes a channel number that indicates a file type other than any of the file types mentioned above, ?TRUNCATE returns error code ERIFT.

Notes

- See the description of ?OPEN in this chapter.

?TUNLOCK
error return
normal return

Input

AC0 One of the following:

- 0 to remove all protection from this ring
- ?TALOCK to remove protection from my ring only
- ?TALOCK+?TMYRING is a no–operation

AC1 Reserved (Set to 0.)

AC2 One of the following:

- Address of a double–word mailbox to contain the output of the previous ?TLOCK or ?TUNLOCK system call
- –1 to indicate that the caller does not want to specify a mailbox
- 0 to indicate that all protection is being removed

Output

AC0 Unchanged

AC1 Unchanged

AC2 Unchanged

Error Codes in AC0

ERICM	Illegal command
ERTLK	No matching ?TLOCK for this ?TUNLOCK
ERVWP	Invalid word pointer passed as a system call argument

Why Use It?

You issue ?TUNLOCK to allow a task–redirection request against a locked task to proceed — provided that the appropriate protection is released.

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

?TUNLOCK removes task redirection protection for the current task in the current ring. It restores redirection protection to a previous state of protection that is equal to or less than the present state of protection. A ?TUNLOCK call basically cancels the effect of an earlier ?TLOCK call on behalf of the same task in the same ring. You should use the output flags from ?TLOCK as input to ?TUNLOCK.

Notes

- See the description of the ?TLOCK system call in this chapter.

?UBLPR — Unblocks a process.

?UBLPR
error return
normal return

Input

AC0	One of the following: • 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
AC2	Reserved (Set to 0.)

Output

AC0	Unchanged
AC1	Unchanged
AC2	Undefined

Error Codes in AC0

ERMPR	System call address error
ERPNM	Illegal process name
ERPRE	Invalid system call input (AC1 is not a 0 or a –1)
ERPRH	Attempt to access a process not in the hierarchy
ERVBP	Invalid byte pointer passed as a system call argument

Why Use It?

?UBLPR is the complement of ?BLKPR; thus, if a process was blocked with ?BLKPR, you must use ?UBLPR to unblock it.

Who Can Use It?

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

What It Does

?UBLPR unblocks the target process that you specify in AC0. Note that the target process must have been blocked with the ?BLKPR system call. (To use ?BLKPR against a process other than the caller's process, Superprocess mode must be turned on.)

?UBLPR lifts the suspension imposed by ?BLKPR on all tasks within the target process. It has no effect upon the state of any other outstanding suspensions against those tasks. However, ?UBLPR only unblocks the target process if it has at least one task that is ready to run. Otherwise, the target process remains blocked until one of its tasks becomes ready to run.

Notes

- See the description of ?BLKPR in this chapter.

?UIDSTAT — Returns the status of a task and an unambiguous identifier.

?UIDSTAT
error return
normal return

Input

AC0	Reserved (Set to 0.)
AC1	One of the following: • TID of the specified task (0 is illegal) • –1 to indicate the current task
AC2	Address of the return packet

Output

AC0	Unchanged
AC1	Unchanged
AC2	Unchanged

Error Codes in AC0

No error codes are currently defined.

Why Use It?

You can use ?UIDSTAT to get a unique task ID, which you can then use to index into multiple–task databases.

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

?UIDSTAT is functionally similar to the ?TIDSTAT and ?IDSTAT system calls; however, unlike ?TIDSTAT and ?IDSTAT, ?UIDSTAT returns a unique TID.

?UIDSTAT accepts a standard TID as input in AC1. However, it also accepts –1 in AC1 to indicate the return of task information for the current task. The operating system passes a word pointer to a return packet in AC2. This packet returns the unique TID, the task status word, the standard TID, and the task's priority. (See Figure 2–233 for the packet structure.)

The unique TID value ranges between 1 and the maximum number of tasks specified by the Ring 7 process.

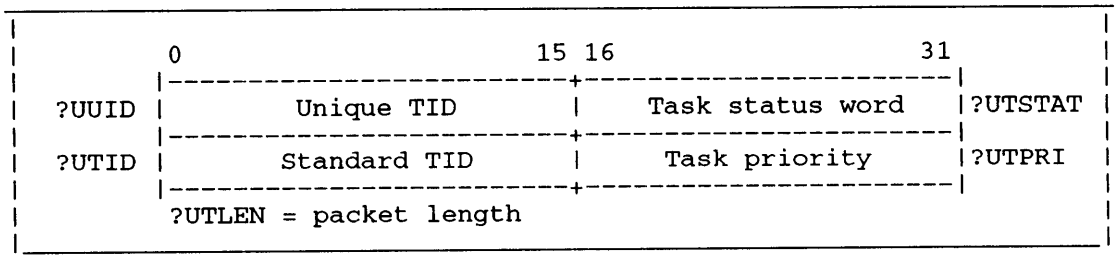


Figure 2–233. Structure of ?UIDSTAT Packet

?UNWIND — Unwinds the stack and restores the previous environment (16–bit processes only).

?UNWIND
normal return

Input

None

Output

AC0 One of the following:

- Base address of the restored resource
- 0

AC1 Return address of the previous resource

AC2 Unchanged

Error Codes in AC0

No error codes are currently defined

Why Use It?

?UNWIND pops the current return block off the stack, just as RTN does, but gives you the option of passing control to an address relative to the return address or to an address relative to the base of the restored resource. Thus, you can use ?UNWIND to restore the stack environment of a previously called resource, without necessarily returning to its RTN address.

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

?UNWIND unwinds the user stack and restores it to the previous environment. As output to ?UNWIND, the operating system returns the base address of the restored resource, immediately following the ?UNWIND; that is, control returns to the calling resource. At this point, the calling resource can transfer control to the return address in AC1, to a fixed address relative to the return address (contents of AC1 + n), or to a fixed address within the previous resource (contents of AC0 + n). If you did not use a resource system call to call the target resource, the operating system returns 0 to AC0.

The ?UNWIND caller must be part of the root resource or part of a resource that will not be released when the stack is unwound. Note that the code generated by ?UNWIND cannot be a single word. (This can affect any skip instructions you issue before you issue ?UNWIND.)

?UNWIRE — Unwires pages previously wired.

?UNWIRE
error return
normal return

Input

AC0	Starting address of the region to unwire
AC1	Ending address of the region to unwire
AC2	Reserved (Set to 0.)

Output

AC0	Unchanged
AC1	Unchanged
AC2	Undefined

Error Codes in AC0

ERICM	Illegal system command (invalid system call for 16–bit processes)
ERPRE	Invalid system call parameter
ERVWP	Illegal system command

Why Use It?

?UNWIRE is the complement of ?WIRE; thus, if a page was wired with ?WIRE, you must use ?UNWIRE to unwire it. You issue one ?UNWIRE to revoke all previous ?WIREs, even though each ?WIRE may be for a different address region.

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

?UNWIRE removes the wired status from a range of addresses in your working set. (A "wired" page is a page that is permanently bound to the working set.) Note that ?UNWIRE can reduce your minimum working set size.

Before you issue ?UNWIRE, load AC0 and AC1 with the starting and ending addresses, respectively, of the region you want to unwire. If the starting and ending addresses fall on page boundaries, ?UNWIRE affects those boundaries and all pages between them. Otherwise, ?UNWIRE affects only the full pages between the starting and ending addresses.

Notes

- See the description of ?WIRE in this chapter.

?UPDATE — Flushes file descriptor information.

?UPDATE
error return
normal return

Input

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

Output

AC0	Undefined
AC1	Unchanged
AC2	Undefined

Error Codes in AC0

ERACU Channel not open
ERCIU Channel in use
ERSIM Simultaneous request on same channel
ER_FS_TLA_MODIFY_VIOLATION
Attempt to modify an AOS/VS II file with ?ODTL value supplied in ?GOPEN packet

Why Use It?

?UPDATE is useful whenever you cannot afford to lose file descriptor information if your system crashes.

Who Can Use It?

There are no special process privileges needed to issue this call. The caller must have obtained a channel number to the file, via ?OPEN or ?GOPEN, before issuing ?UPDATE. Also, the caller must have had Read or Write access to the file at the time of the ?OPEN or ?GOPEN call.

What It Does

?UPDATE flushes any memory–resident file descriptor information for the target file to disk, and thereby updates it. File descriptor information includes user data areas (UDAs). The file you update must be a user file; if you attempt to update a system file, ?UPDATE takes the normal return, but has no effect.

?UPDATE guarantees the integrity of all the data you previously wrote to the file via a ?FLUSH system call. Some data, however, such as shared pages and any data buffered by the Agent, may not have been written to disk yet. To flush this data, use the ?ESFF system call or the ?WRITE system call with the force output bit set.

Notes

- See the descriptions of ?WRITE, ?FLUSH, ?ESFF, and ?RPAGE in this chapter.

?VALAD — Validates a logical address.

?VALAD
error return
normal return

Input

AC0	Logical address to validate
AC1	Type of validation
AC2	Reserved (Set to 0.)

Output

AC0	Unchanged
AC1	Indicator of which validations succeeded
AC2	Undefined

Error Codes in AC0 and AC1

ERACC	Access denied

Why Use It?

?VALAD allows the caller to verify that an address is valid or that access is available to that address.

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

?VALAD validates a logical address for a particular type of access. You can check ?RDAC (read), ?WRAC (write), ?EXAC (execute), and ?VLAC (validate) access. Also, you can check any combination of access types at one time.

?VALAD takes the error return if any of the requested accesses being checked fails. To indicate which accesses succeeded, the operating system returns the ERACC (access denied) error code in AC0 and AC1.

The logical AND of the access privileges that you want to check and the actual privileges that the address has always return in AC1. Therefore, if the operating system takes a normal return, AC1 appears unchanged. Every privilege does not have to fail for the operating system to take the error return. If there is an error, the privileges that succeed are indicated in AC1.

NOTE: To use ?VALAD optimally, the server must perform the following checks before it issues this system call, where PTE = page table entry:

1. Check whether the address is being passed below the ring in which the caller resides (that is, is it a Trojan pointer). If so, fail. If not, succeed.

2. Do a LPHY on the pertinent logical address. If the LPHY succeeds, examine the PTE for the appropriate access. If the PTE does not have the appropriate access, fail. If the PTE does have the appropriate access, succeed.

?VALIDATE — Validates an area for Read or Write access.

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

Input

AC0	Unused
AC1	Unused
AC2	Address of packet

Output

AC0	Unchanged
AC1	Unchanged
AC2	Unchanged

Error Codes in AC0

(See also the section "Error Conditions.")

ERACC	Access denied
ERICM	Illegal system command
ERPRE	Invalid system call parameter
ERUFR	Unknown function request
ERVWP	Invalid address passed as system call argument

Why Use It?

?VALIDATE is most useful to inner ring server development. Using the ?VALIDATE call, servers can avoid user traps caused by referencing an invalid area passed from the customer program. See examples in the later section "Examples of Use."

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

?VALIDATE provides the caller with an interface to the Agent validation routines. An area is defined by a word pointer and word length, or by a byte pointer and byte length. Areas can be validated for Read or Write access. Also, there is a validation function for validating strings of unknown length but known maximum length, using a user–specified delimiter table.

Packet Format

The five functions offered by this system call share the same packet format. The type of processing which will occur depends on the contents of offset ?VFUNC. Table 2–205 explains the functions and their codes.

Table 2–205. ?VALIDATE Functions and Their Codes

Offset	Description
?VFUNC	FUNCTION CODES and FLAG ?RVWPL--validate an area defined by word pointer and word length for Read access. ?WVWPL--validate an area defined by word pointer and word length for Write access. ?RVBPL--validate an area defined by byte pointer and byte length for Read access ?WVBPL--validate an area defined by byte pointer and byte length for Write access. ?RVBPX--validate an area defined by byte pointer and maximum byte length (unknown actual length) for Read access, using a delimiter table. ?VRNBR--If this flag is set to 1, the ring in offset ?VRING will be interpreted as a ring number (value 4 - 7). If this flag is set to 0, ?VRING will be interpreted as a ring field in bits 1S1-1S3 (value 0;4 to 0;7).
?VRES	Reserved. (Set to 0.)
?VPOINTER (double-word)	Starting address of the area to be validated. This will be interpreted as a word address for ?RVWPL and ?WVWPL, and as a byte address for the other functions.
?VRING (double-word)	Ring to be used for validation purposes. See ?VRNBR flag above for interpretation of this offset.
?VLENGTH (double-word)	Length of the area to be validated. This will be interpreted as a word length for ?RVWPL and ?WVWPL, and as a byte length for the other functions.
?VDELIM (double-word)	Word address of a 16-bit delimiter table to be used in validating areas of unknown actual length by the validation function ?RVBPX. You may also specify (-1) to designate the default delimiters (null, New Line, and form feed -- ASCII 0, 12, or 14). This parameter is ignored by the other validation functions.
?VERROR (double-word)	Returns zero if the area was valid for the type of access specified. If this value is not zero, the area was invalid (see "Other Notes", next page).
?VRESD (double-word)	Reserved for expansion. (Set to 0.)

The packet length is defined by the parameter ?VPLTH.

Error Conditions

The exception return for this call will be taken only if the call is invalid for one of the following reasons:

- It was called by a 16–bit process in Ring 7. ERICM — "Invalid System Command" — will indicate this error.
- The packet address in AC2 is invalid. ERVWP will be returned.
- The function code in ?VFUNC of the packet is unknown. ERUFR will be returned in this case.
- For function ?RVBPX only, if the contents of ?VDELIM is not (–1) or a valid address, error code ERVWP will be returned.
- The ring specified in ?VRING is invalid. ERVPW will be returned.

Note that the normal return indicates that the call was made correctly. However, it does NOT indicate the validity of the area. You must check the error code returned by ?VERROR, which will be zero if the type of access specified is possible. If ?VERROR is nonzero, it will be one of the following codes:

ERVWP — This code is returned for functions ?RVWPL or ?WVWPL if the specified area is invalid for the indicated type of access.

ERVBP — This code is returned for functions ?RVBPL, ?WVBPL, or ?RVBPX if the specified area is invalid for the indicated access type.

ERPRE — will be reported for the following error conditions:

- The area length specified by ?VLEN is invalid. The area length must be a positive number which does not exceed the number of words or bytes in one ring.

Notes

- An area will be considered invalid if it crosses the unshared/shared boundary.

Examples of Use

Suppose an inner ring server for a database system gets a request to return some information from a database. The database caller, a Ring 7 application in this example, passes a byte address and byte length to specify the receiving area.

The database server may attempt to write the information into the customer's space without validation; however, it risks causing a USER TRAP from its ring. By calling ?VALIDATE with Ring 7 in ?VRING, and the user's byte address and length in ?VPTR and ?VLEN, the database server can simply return an error code for the request.

Suppose a Ring 7 caller provides a Ring 4 address to a Ring 4 server. ?VALIDATE should be set up to indicate that the caller's ring is 7; this "ring maximization" checking will return a validation error to protect the server.

Another example: Consider the situation where a string parameter, such as a filename, is an argument to some service. It is reasonable for the user to put the filename at the top of his unshared area, such that the string would be considered invalid if the entire length of a filename was used with function ?RVBPL. The user may have a null–terminated string "FOO<0>" which fits in this area. By using the function ?RVBPX and indicating the default delimiters table, the server can easily check the validity of this situation.

?VCUST — Verifies a customer in Ring 7.

?VCUST
error return
normal return

Input

AC0	PID of the process to verify
AC1	Reserved (Set to 0.)
AC2	Reserved (Set to 0.)

Output

AC0	Unchanged
AC1	Undefined
AC2	Unchanged

Error Codes in AC0

ERCBK	Connection broken
ERCDE	Connection does not exist
ERNAS	Process is not a server
ERRNI	Invalid ring number

Why Use It?

?VCUST gives you access to the system's connection table to find out whether a segment image of a process is a customer of Ring 7. ?VCUST is also a useful way to find out if a customer process has broken the connection from its end.

Who Can Use It?

There are no special process privileges needed to issue this call, although you must be a server process. There are no restrictions concerning file access.

What It Does

?VCUST tests to see whether a segment image of the Ring 7 process that you specify in AC0 is a customer of the calling segment image.

If the target process in Ring 7 is the caller's customer and there is an unbroken connection between the two, control takes the normal return. If the target process in Ring 7 is not the caller's customer, ?VCUST fails on error code ERCDE. If the target process in Ring 7 is a customer, but the connection has been broken (for example, the customer issued a ?DCON system call), ?VCUST fails on error code ERCBK.

?VMEM — Changes the partition size of a process.

AOS/RT32 only

?VMEM *[packet address]*
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 direct the action to the calling process
AC1	One of the following: • –1 if AC0 contains a byte pointer • 0 if AC0 contains a PID • 0 if AC0 contains –1; the OS ignores AC1 in this case
AC2	Address of the ?VMEM packet, unless you specify the address as an argument to ?VMEM

Output

AC0	Unchanged
AC1	Unchanged
AC2	Address of the ?VMEM packet

Error Codes in AC0

ERAWM	Too many pages are wired
ERMEM	Insufficient memory is available

Why Use It?

Issue ?VMEM to modify the working set size of a process's unshared partition or to modify the size of the global shared partition. Changing the partition size makes a process run faster or slower.

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

?VMEM changes the partition size of any process on the system. You use AC0 and AC1 to specify the process.

Figure 2–234 shows the structure of the ?VMEM parameter packet, and Table 2–206 describes its contents. The offsets of this parameter packet do not appear in any DG–supplied parameter file. To use this packet you'll have to define your own parameter file, or hard code the values in Figure 2–234 into your program.

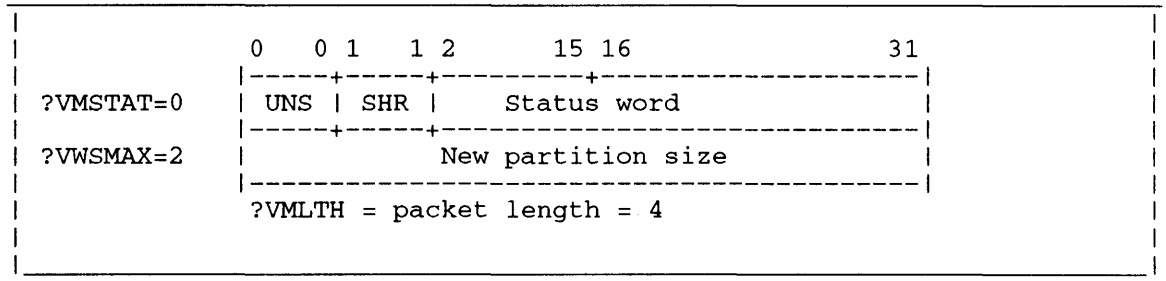


Figure 2–234. Structure of ?VMEM Packet

Table 2–206. Contents of ?VMEM Packet

Offset	Contents
?VMSTAT (doubleword)	Status word. Supply 0 in Bits 2 through 15. If you supply 10 in Bits 0 and 1, then the OS modifies the unshared partition size of the process you specify in AC0 and AC1. If you supply 01 in Bits 0 and 1, then the OS modifies the global shared partition and ignores the contents of AC0 and AC1.
?VWSMAX (doubleword)	Supply the new size of the partition in words as a 32-bit unsigned integer.

NOTE: System call ?VMEM does not appear in any DG–supplied parameter file. You must hard code the following calling sequence to issue ?VMEM since the regular calling sequence does not work for ?VMEM.

```
LJSR     @6
.WORD    117
<error return>
<normal return>
```

?VRCUST — Verifies a customer in a specified ring.

?VRCUST
error return
normal return

Input

AC0	PID of the process to verify
AC1	Reserved (Set to 0.)
AC2	Bits 0 through 28 are reserved (Set to 0.) Bits 29 through 31 contain the ring number of the target in the specified process

Output

AC0	Unchanged
AC1	Undefined
AC2	Unchanged

Error Codes in AC0

ERCBK	Connection broken
ERCDE	Connection does not exist
ERNAS	Process is not a server
ERRNI	Invalid ring number

Why Use It?

?VRCUST gives you access to the system's connection table to find out whether a segment image of a process is a customer of the calling process ring. ?VRCUST is also a useful way to find out if a customer process has broken the connection from its end.

Who Can Use It?

There are no special process privileges needed to issue this call, although you must be a server process. There are no restrictions concerning file access.

What It Does

?VRCUST tests to see whether a segment image of the process that you specify in AC0 is a customer of the calling segment image.

If the target process ring is the caller's customer and there is an unbroken connection between the two, control takes the normal return. If the target process ring is not the caller's customer, ?VRCUST fails on error code ERCDE. If the target process ring is a customer, but the connection has been broken (for example, the customer issued a ?DCON system call), ?VRCUST fails on error code ERCBK.

?VTFCREATE — Creates a Virtual Timer Facility timer.

AOS/RT32 only

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

Input

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

Output

AC0	Unchanged
AC1	Unchanged
AC2	Address of the ?VTFCREATE packet

Error Codes in AC0

ERADR	Illegal starting address
ERPRE	Illegal system call parameter
ERTID	Task ID error
ERVIL	Illegal VTF parameter
ERVMO	Illegal VTF mode
ERVNA	VTF timer not available
ERVNG	Negative VTF setpoint not allowed
ERVOR	VTF setpoint overflow

Why Use It?

Issue ?VTFCREATE to set up the periodic scheduling of events through the Virtual Timer Facility (VTF). This system call defines a time interval and an action to take at each interval.

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

?VTFCREATE creates a virtual timer under the Virtual Timer Facility (VTF). Virtual timers provide a periodic event at the time interval you have specified in the system call's parameter packet. The VTF dispatcher, which is part of AOS/RT32, monitors the VTF timers and performs the actions you've specified when events occur. The dispatcher assigns priorities to the events and makes them interruptible so that the highest priority event is always handled immediately, even if a low–priority VTF event is in progress.

The dispatcher can detect timing overruns when an event's action is not complete before the next occurrence of the event. You specify the actions to take when the dispatcher detects timing overruns. A timing overrun occurs when, after ?VTFCREATE starts a timer, the timer calls for an event before the previous event for that same timer has completed. ?VTFCREATE and ?VTFMODIFY specify an action to take when the dispatcher later detects an overrun, but these two system calls themselves cannot detect an overrun.

The VTF dispatcher runs as part of the interrupt service routine (ISR) for the physical timer interrupt. The time base of the new VTF timer is expressed in units that are the number of physical timer *ticks* — unless the timer is *cascaded*. Cascaded timers are expressed in units of the ticks of the timer from which they are cascaded. See the section "Timer Setting Examples" at the end of this description of ?VTFCREATE.

There are two types of actions: executing a routine you have written at interrupt level and sending a signal to a waiting base–level task that system call ?WTVSIG has suspended. Interrupt–level routines provide fully deterministic and repeatable responses, along with low overhead, but these routines may issue only those system calls that are legal at the interrupt level. While you completely define the interrupt–level routine, the operating system places in AC0 the number of missed timer events since the last execution of the interrupt–level routine.

On the other hand, sending signals to waiting base–level tasks allows periodic communication to tasks running with full operating system facilities. However, these signals involve the additional overhead of normal scheduling and context switching.

When ?VTFCREATE executes it returns, in offset ?VTFID of its packet, a unique identifying number for the timer it has just created. You place this number in the packets of other VTF–related system calls to identify the newly created VTF timer.

Figure 2–235 shows the structure of the ?VTFCREATE parameter packet, and Table 2–207 describes its contents.

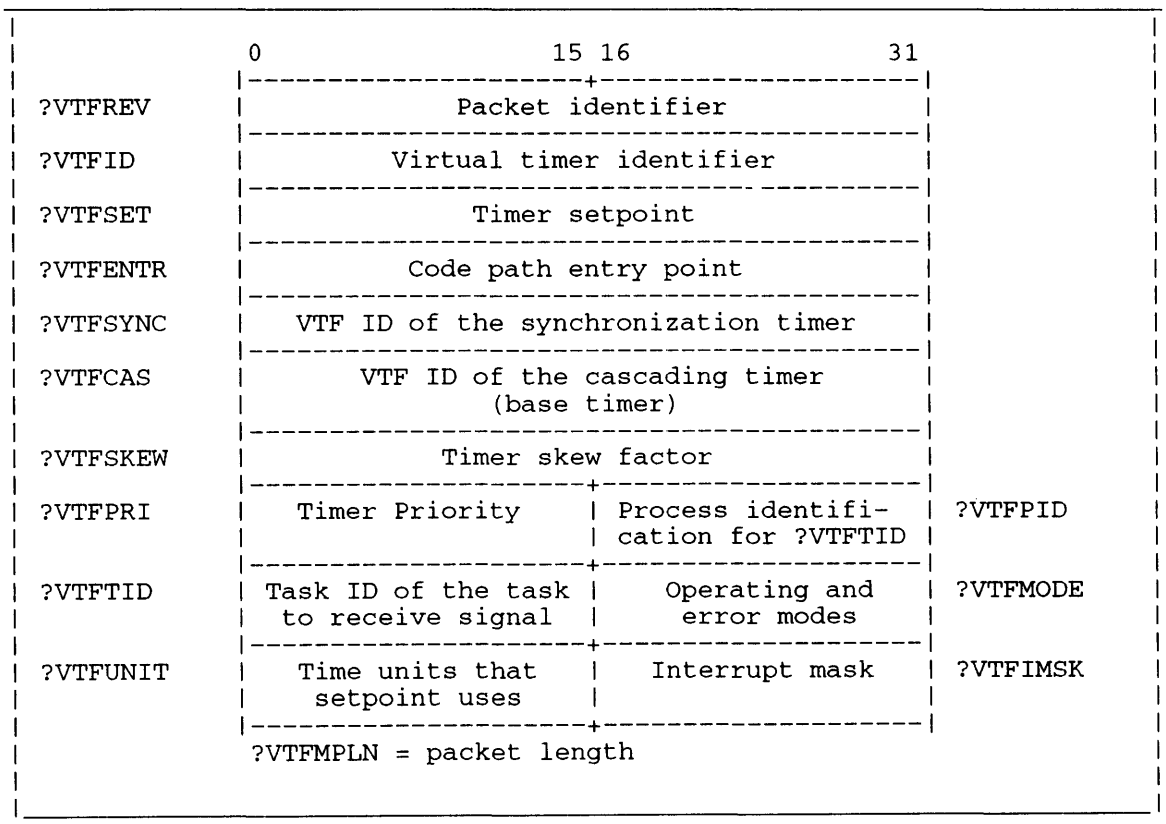


Figure 2–235. Structure of ?VTFCREATE Packet

Table 2–207. Contents of ?VTFCREATE Packet

Offset	Contents
?VTFREV (doubleword)	Packet identifier. Place ?VINIREV here.
?VTFID (doubleword)	Place 0 here. The OS returns the VTF identifier of the newly created timer.
?VTFSET (doubleword)	Place the time period of the timer here. Use either the time units that offset ?VTFUNIT sets or else the number of "ticks" of the base timer if the timer is cascaded. The value must be greater than 0.
?VTFENTR (doubleword)	Place the starting address of the code you want executed at each event. This address must be in Ring 7 and the code path may not cross to any other ring. Place 0 here if offset ?VTFMODE does not contain the value ?VMODILV.
?VTFSYNC (doubleword)	Supply the VTF ID of the synchronization timer. If there is no such timer, supply 0. In this case there is no synchronization and the timer that ?VTFCREATE is about to create will start immediately.
?VTFCAS (doubleword)	Supply the VTF ID of the cascading timer. If there is no such timer, supply 0. In this case there is no cascading and the timer that ?VTFCREATE is about to create will run off the physical clock.
?VTFSKEW (doubleword)	Supply the number of time units, as specified in offset ?VTFUNIT, by which to offset the initial timed event from the timer event that offset ?VTFSYNC specifies. If offset ?VTFSYNC contains 0, then this offset must also contain 0.
?VTFPRI	Supply a 16-bit unsigned number as the priority of the new timer. 1 specifies the highest priority.
?VTFPID	Supply the process ID of the task that offset ?VTFTID specifies. The default value of -1 specifies the calling task's process ID.
?VTFTID	Supply the task ID of the task you want to receive a signal at each event. You must supply 0 if offset ?VTFMODE does not contain the value ?VMODSIG. The default value is -1 to have the calling task receive a signal at each event. Also, ?VTFCREATE returns error code ERTID if the ID belongs to a task that is already the target of another VTF timer signal.

(continued)

Table 2–207. Contents of ?VTFCREATE Packet

Offset	Contents
?VTFMODE	This mode word has the following possible values and effects. ?VMODILV Interrupt level mode. The action is to run the code path whose starting address is in offset ?VTFENTR. The code path must be in the logical address space of the process whose ID is in offset ?UTFPID. ?VMODSIG Signaling mode. The action is to send a VTF signal to the task specified in offsets ?VTFTID and ?VTFPID. ?VMODNUL Null mode. No action is taken. Use null mode when a timer's only function is to be a cascade base timer for other timers. NOTE: Specify exactly one of the previous three values. ?VERRIGN Ignore overrun errors. ?VERRTRM Suspend the timer immediately when an overrun error occurs and send an error message to the VTF error queue. ?VERRNTR Suspend the timer after receiving <n> consecutive overrun errors; <n> is the value you supply in Bits 8-15 of offset ?VTFMODE. NOTE: Specify exactly one of the previous three values. ?VTFINISUS Suspend the timer initially. This achieves precise starting times based on later events. ?VTFONESHOT The timer runs in one-shot mode by suspending itself after its first event. Use this mode to produce precise time-delayed events from interrupt or base-level code along with system call ?VTFSUS.

(continued)

Table 2–207. Contents of ?VTFCREATE Packet

Offset	Contents
?VTFMODE (continued)	?VTFSAVE This has the VTF handler save your stacks before executing the interrupt-level code. The default action is not to save your stacks. Supply this value if you use stacks and need to recover from a system hang (See the explanation of system call ?WTVERR.). Then, AOS/RT32 saves the four wide stack pointers and the wide stack fault handler.
?VTFUNIT	Supply the time units that the setpoint (offset ?VTFSET) and the skew factor (offset ?VTFSKEW) are expressed in. The OS ignores this value if offset ?VTFCAS does not contain 0 or if the VTF is currently attached to a user interrupt via system call ?FIDEF. The legal units are o ?VTFUSEC -- microseconds (0.000001 second) o ?VTFSEC -- seconds o ?VTFTICK -- the frequency of the physical timer The OS rounds the setpoint to the nearest multiple of the base frequency of the physical timer. The OS returns error code ERVOR if the the value of the setpoint results in more "ticks" of the physical timer than a 32-bit unsigned number can contain.
?VTFIMSK	Supply the interrupt mask that you want the OS to apply at the start of an interrupt-level code path. If you don't supply the value ?VMODILV in offset ?VTFMODE, then supply 0 here.
?VTFMPLN	Packet length.

(concluded)

Timer Setting Examples

If the physical timer is running at 1000 hertz (1000 cycles per second = once every millisecond) and the newly created VTF timer specifies a time base setpoint of 20, then the associated periodic event will occur every 20 milliseconds — a rate of 50 hertz = 50 cycles per second.

If you specify a cascaded timer, it uses the cascading timer's time base as the basis of its (the first timer's) setpoint. For example, assume that your new timer specifies timer 13 as its cascading timer and that timer 13 is running at 100 hertz (= 10 millisecond time base). If your new timer has a setpoint of 25, then it runs once each 25 times that the timer 13 event occurs. This number is (10 milliseconds)*25, which is a 250 millisecond (= 0.25 second) period for a frequency of 4 hertz.

You can cascade a virtual timer to any level simply by specifying an already cascaded timer as the cascading timer. However, the cascading timer you specify must already be initialized or else the OS will return an error.

NOTE: The OS suspends or deletes cascaded timers if the timer from which they are cascaded is suspended or deleted, respectively. When this cascade timer (i.e., source timer) restarts, the cascaded timers automatically restart. The OS returns error ERVNS if you attempt to restart a timer that is suspended solely because its cascade timer is suspended.

Timer Synchronization

Although cascaded timers offer a powerful means of synchronizing timed events to a master clock, occasions can arise when you require more flexible synchronization. For example, you might need to have two events running every 20 milliseconds, but they must be exactly 10 milliseconds apart. In this case you use the synchronization offsets of the packet for ?VTFCREATE to specify another timer to synchronize from with a *skew* factor. Doing this is *not* cascading since the other timer serves only to start the new timer running at a predictable time. The new timer will still run with the time base that its setpoint and cascade timer specify. You can use any existing timer to synchronize from, regardless of its time base.

When the synchronization timer offset, ?VTFSYNC, is not zero, it contains the VTF ID of the synchronizing timer and offset ?VTFSKEW contains the skew count. These offsets delay the actual start of timing for the new timer until the next time the synchronizing timer either times out or registers an event. At this point the OS starts the new timer — but with a setpoint that is the sum of the specified setpoint (offset ?VTFSET contents) and the skew factor (offset ?VTFSKEW contents). If this sum exceeds the maximum 32–bit number, the setpoint becomes the maximum value. This is true whenever the new timer starts running; the result is an initially longer timer period for the new timer with its events offset from the events of the synchronization timer. Subsequent time periods do *not* add in the skew factor — only the first time period adds in the skew factor whenever a timer is started.

NOTE: When a timer restarts after being explicitly suspended, it synchronizes with whatever synchronizing timer you specified for it. This synchronization does not happen if the restarting timer was implicitly suspended because it is cascaded and its base timer was suspended. The OS allows deletion of timers that were used as synchronizing timers. In this case the remaining timers that were synchronized no longer are, and the OS immediately starts the remaining timers that were suspended while waiting for the synchronizing timer. You can change the selection of a synchronizing timer by issuing system call ?VTFMODIFY.

You may specify any timer as the synchronizing timer. It generally makes sense to specify one only if the new timer is running at the same frequency as, or an integral multiple of, the synchronizing timer's frequency.

Example of a Synchronization Timer

For example, assume that

- The synchronization timer is running at 50 hertz (once every 20 milliseconds).
- The new timer has a setpoint of 10 (100 hertz = once every 10 milliseconds).
- The new timer has a skew factor of 5.
- The physical clock is running at 1000 hertz (once every millisecond).

Under these conditions the new timer does not start until the synchronization timer event occurs. The new timer's first event occurs 15 milliseconds later with all of its subsequent events occurring at 10 millisecond intervals. The net effect is that the new timer's events will always occur 5 milliseconds before and after the synchronization timer's events, even though the new timer's time base is a multiple of the synchronization timer's time base. Figure 2–236 summarizes this example. In the figure

- 1 indicates a timer number 1 event.
- 2 indicates a timer number 2 event.
- Timer number 1 is the synchronizing timer and runs at 50 hertz = once every 20 milliseconds.
- Timer number 2 is the new timer and runs at 100 hertz = once every 10 milliseconds with a skew factor of 5 milliseconds.

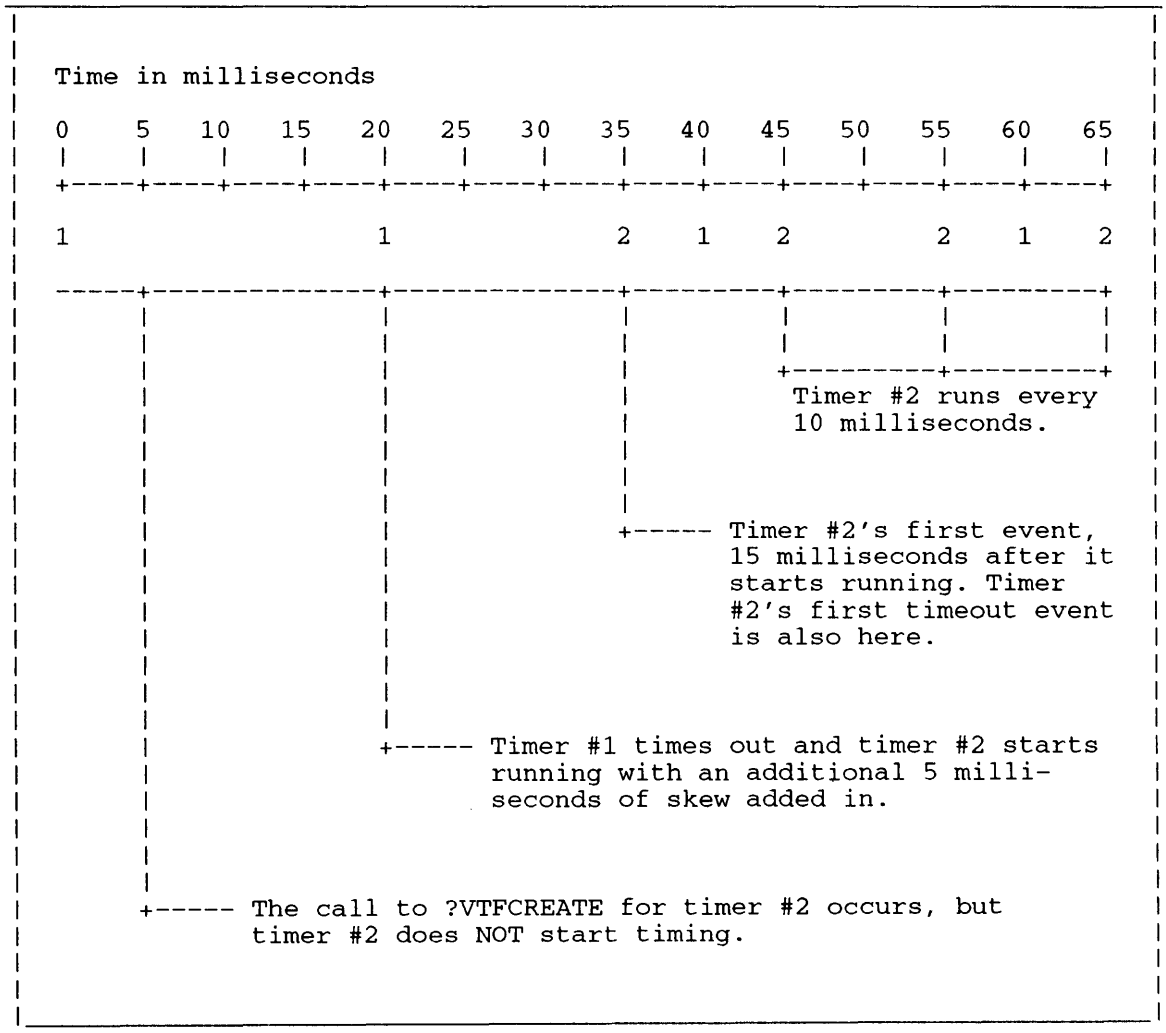


Figure 2–236. Execution of Two Virtual Timers

Notes

- See the descriptions of ?VTFKILL, ?VTFMODIFY, ?VTFSUS, ?WTVERR, and ?WTVSIG in this chapter.

AOS/RT32 only

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

Input

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

Output

AC0	Unchanged
AC1	Unchanged
AC2	Address of the ?VTFKILL packet

Error Codes in AC0

ERVTI	VTF timer ID does not exist

Why Use It?

Issue ?VTFKILL to delete a periodic timer that system call ?VTFCREATE previously created.

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

?VTFKILL, and only ?VTFKILL, lets you delete a previously created Virtual Timer Facility (VTF) timer.

Figure 2–237 shows the structure of the ?VTFKILL parameter packet, and Table 2–208 describes its contents.

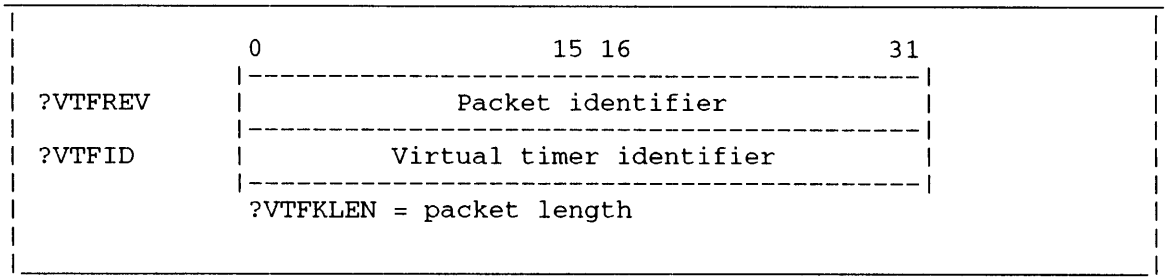


Figure 2–237. Structure of ?VTFKILL Packet

Table 2–208. Contents of ?VTFKILL Packet

Offset	Contents
?VTFREV (doubleword)	Packet identifier. Place ?VTFKREV here.
?VTFID (doubleword)	Place the VTF identifier here of the existing timer that you want to kill. The OS returns an error if the timer does not exist.

?VTFMODIFY — Modifies a Virtual Timer Facility timer.

AOS/RT32 only

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

Input

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

Output

AC0	Unchanged
AC1	Unchanged
AC2	Address of the ?VTFMODIFY packet

Error Codes in AC0

ERADR	Illegal starting address
ERPRE	Illegal system call parameter
ERTID	Task ID error
ERVIL	Illegal VTF parameter
ERVMO	Illegal VTF mode
ERVNA	VTF timer not available
ERVNG	Negative VTF setpoint not allowed
ERVOR	VTF setpoint overflow
ERVTI	VTF timer ID does not exist

Why Use It?

Issue ?VTFMODIFY to modify the periodic scheduling of events through the Virtual Timer Facility (VTF). System call ?VTFCREATE creates this periodic scheduling.

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

?VTFMODIFY allows you to change the parameters of a VTF timer without deleting the timer. This means you can change the iteration rate of the timer, the priority of a timer, which task the timer signals, and the code path the timer executes without suspending current monitoring or control functions.

?VTFMODIFY uses a packet that is structurally identical to ?VTFCREATE's packet. However, there are two differences between the packets:

- Offset ?VTFREV receives different values that identify the packet revision number.
- Offset ?VTFID either receives a value from you or else returns one to you. You supply a number in ?VTFID that specifies a timer you want to modify via ?VTFMODIFY; you receive a number in ?VTFID that identifies a timer ?VTFCREATE has just created.

This closeness between packets lets you efficiently modify timers by changing only the offsets of ?VTFCREATE's packet and issuing ?VTFMODIFY with this packet. In other words, you don't have to build a separate packet for ?VTFMODIFY — as long as you change the value of offset ?VTFREV and interpret differently the value of offset ?VTFID.

?VTFMODIFY lets you select, via Bit ?VTFMODSUS in offset ?VTFMODE, whether the parameters change with a brief suspension or with no suspension. In the former case the timer briefly suspends while all the modifiable parameters are updated so that the timer doesn't run with its parameters only partially updated. In the latter case the timer can run with momentarily inconsistent parameters if more than one of the modifiable parameters differs from its previous value. We recommend, but don't require, that you set Bit ?VTFMODSUS when you change more than one parameter. Otherwise, we cannot guarantee the order of parameter changing.

Figure 2–238 shows the structure of the ?VTFMODIFY parameter packet, and Table 2–209 describes its contents.

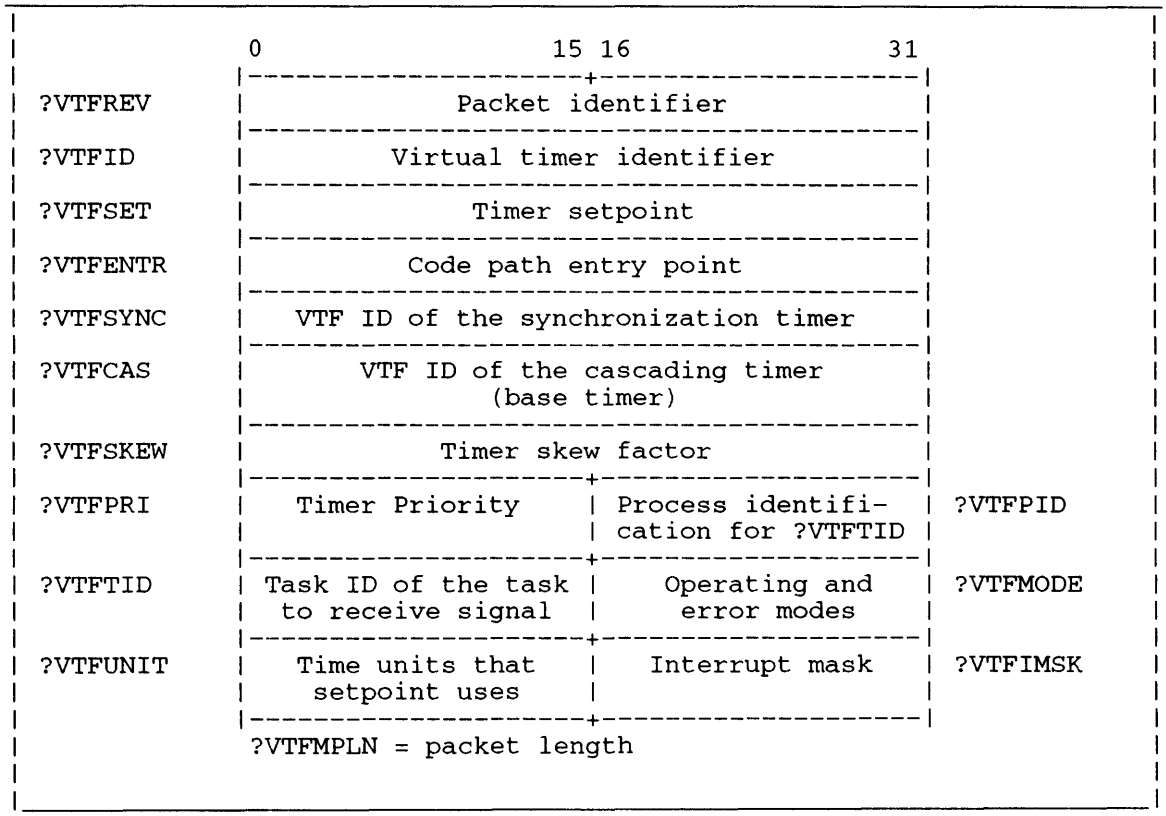


Figure 2–238. Structure of ?VTFMODIFY Packet

Table 2-209. Contents of ?VTFMODIFY Packet

Offset	Contents
?VTFREV	Packet identifier. Place ?VTFMREV here.
?VTFID	Place the VTF identifier here of the timer that you want to modify. The OS returns an error if the timer does not exist.
?VTFSET	Place the time period of the timer here. Use either the time units that offset ?VTFUNIT sets or else the number of "ticks" of the base timer if the timer is cascaded. The value must be greater than 0.
?VTFENTR	Place the starting address of the code you want executed at each event. This address must be in Ring 7 and the code path may not cross to any other ring. Place 0 here if offset ?VTFMODE does not contain the value ?VMODILV. The entry point is in the logical address space of the process specified in offset ?VTFPID.
?VTFSYNC	Supply the VTF ID of the synchronization timer. If there is no such timer, supply 0. In this case there is no synchronization and the timer that ?VTFMODIFY is about to modify will start immediately. The OS returns an error if the timer doesn't exist.
?VTFCAS	Supply the VTF ID of the cascading timer. If there is no such timer, supply 0. In this case there is no cascading and the timer that ?VTFMODIFY is about to modify will run off the physical clock. Note that if you are changing this offset the timer, and all timers cascaded from it, are briefly suspended regardless of the value of Bit ?VTFMODSUS.
?VTFSKEW	Supply the number of time units, as specified in offset ?VTFUNIT, by which to offset the initial timed event from the timer event that offset ?VTFSYNC specifies. If offset ?VTFSYNC contains 0, then this offset must also contain 0.
?VTFPRI	Supply a 16-bit unsigned number as the priority of the new timer. 1 specifies the highest priority.
?VTFPID	Supply the process ID of the task that offset ?VTFTID specifies. The default value of -1 specifies the calling task's process ID.
?VTFTID	Supply the task ID of the task you want to receive a signal at each event. You must supply 0 if offset ?VTFMODE does not contain the value ?VMODSIG. The default value is -1 to have the calling task receive a signal at each event. Also, ?VTFMODIFY returns error code ERTID if the ID belongs to a task that is already the target of another VTF timer signal.

(continued)

Table 2–209. Contents of ?VTFMODIFY Packet

Offset	Contents
?VTFMODE	This mode word has the following possible values and effects. ?VMODILV Interrupt level mode. The action is to run the code path whose starting address is in offset ?VTFENTR. The code path must be in the logical address space of the process whose ID is in offset ?UTFPID. ?VMODSIG Signaling mode. The action is to send a VTF signal to the task specified in offsets ?VTFTID and ?VTFPID. ?VMODNUL Null mode. No action is taken. Use null mode when a timer's only function is to be a cascade base timer for other timers. NOTE: Specify exactly one of the previous three values. ?VERRIGN Ignore overrun errors. ?VERRTRM Suspend the timer immediately when an overrun error occurs and send an error message to the VTF error queue. ?VERRNTR Suspend the timer after receiving <n> consecutive overrun errors; <n> is the value you supply in Bits 8-15 of offset ?VTFMODE. NOTE: Specify exactly one of the previous three values. ?VTFONESHOT The timer runs in one-shot mode by suspending itself after its first event. Use this mode to produce precise time-delayed events from interrupt or base-level code along with system call ?VTFSUS. ?VTFSAVE This has the VTF handler save your stacks before executing the interrupt-level code. The default action is not to save your stacks. Supply this value if you use stacks and need to recover from a system hang (See the explanation of system call ?WTVERR.). Then, AOS/RT32 saves the four wide stack pointers and the wide stack fault handler. ?VTFMODSUS Supply this value to suspend the timer while the modification occurs. Otherwise, the timer continues while the OS modifies it.

(continued)

Table 2–209. Contents of ?VTFMODIFY Packet

Offset	Contents
?VTFUNIT	Supply the time units that the setpoint (offset ?VTFSET) and the skew factor (offset ?VTFSKEW) are expressed in. The OS ignores this value if offset ?VTFCAS does not contain 0 or if the VTF is currently attached to a user interrupt via system call ?FIDEF. The legal units are o ?VTFUSEC -- microseconds (0.000001 second) o ?VTFSEC -- seconds o ?VTFTICK -- the frequency of the physical timer The OS rounds the setpoint to the nearest multiple of the base frequency of the physical timer. The OS returns error code ERVOR if the the value of the setpoint results in more "ticks" of the physical timer than a 32-bit unsigned number can contain.
?VTFIMSK	Supply the interrupt mask that you want the OS to apply at the start of an interrupt-level code path. If you don't supply the value ?VMODILV in offset ?VTFMODE, then supply 0 here.

(concluded)

NOTE: If you set Bit ?VTFMODSUS, then the OS briefly suspends the timer (for an indeterminate length of time) to ensure the consistent updating of more than one parameter. This suspension is *independent* of any suspension that system call ?VTFSUS or overrun errors cause.

Notes

- See the descriptions of ?VTFCREATE, ?VTFKILL, ?VTFSUS, ?WTVERR, and ?WTVSIG in this chapter.

?VTFSUS Suspends or Restarts a Virtual Timer Facility timer.

AOS/RT32 only

?VTFSUS
error return
normal return

Input

AC0	Reserved (Set to 0.)
AC1	VTF ID
AC2	Function code

Output

AC0	Unchanged
AC1	Unchanged
AC2	Unchanged

Error Codes in AC0

ERVFU	Illegal VTFSUS function code
ERVNR	VTF timer not running when suspended
ERVNS	VTF timer not suspended when restarted
ERVTI	VTF timer ID does not exist

Why Use It?

Issue ?VTFSUS to prevent a VTF timer from causing more events or to restart any timer previously suspended via ?VTFSUS.

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 possible functions that you can choose by placing one of two codes in AC2. The two codes and their corresponding functions are as follows.

?VTFSUD	Suspend a running VTF timer
?VTFRST	Restart a suspended VTF timer

Once a timer has started you can suspend it by issuing ?VTFSUS with function code ?VTFSUD. The OS stops the timer, which in turn stops counting clock ticks — but the OS does not clear the clock tick count. No more periodic events occur until you restart the timer. A suspended timer is in the same state that any timer (call it A) is in just after system call ?VTFCREATE creates A but before A's synchronization timer, specified to ?VTFCREATE, starts A.

If the OS detects an overrun error, you can suspend the overrunning timer depending on the overrun error option you specify in offset ?VTFMODE of system call ?VTFCREATE's packet. The OS returns error ERVNR for ?VTFSUS if you issue the system call against a timer that a previous issuance of ?VTFSUS suspended.

NOTE: The OS does *not* delete a suspended timer. The timer still exists with all its parameters intact, and you can restart it at any time.

The only way you can restart a suspended timer is by issuing ?VTFSUS with function code ?VTFRST. As soon as the OS processes this call, the timer starts counting down clock ticks from the point it left off when you previously suspended it. The OS returns error ERVNS if the timer you specify is not a suspended timer.

Example

You suspend a timer when it has 5 seconds left before its next timeout. 20 seconds later you restart the timer. Then, the timer times out 5 seconds later (or 25 seconds after the original suspension).

Notes

- You can issue ?VTFSUS from interrupt level.
- VTF timers suspend themselves when they detect an overrun error. This suspension is the same as a ?VTFSUS suspension, and you can then issue ?VTFSUS with the ?VTFRST function code.
- See the description of ?VTFCREATE in this chapter, particularly its section about the overrun error handling options.

?VTFXIT Exits from a Virtual Timer Facility interrupt routine.

AOS/RT32 only

?VTFXIT
error return
normal return

Input

AC0	Reserved (Set to 0.)
AC1	Reserved (Set to 0.)
AC2	Reserved (Set to 0.)

Output

AC0	Unchanged
AC1	Unchanged
AC2	Unchanged

Error Codes in AC0

This system call returns no error codes.

Why Use It?

You must issue ?VTFXIT to return from the interrupt–level routine that the VTF's dispatcher invoked.

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 your interrupt–level code to the VTF's dispatcher. Upon this return the dispatcher checks for any VTF events that might have occurred since the execution of your interrupt–level code began. If any VTF events have occurred, the OS invokes their actions. Otherwise, the OS dismisses the current timer interrupt and base level processing continues from the point of interruption. Note that ?VTFXIT does not force base level task rescheduling.

Notes

- See the description of ?VTFCREATE in this chapter.

?WALKBACK — Returns information about previous frames in the stack (16–bit processes only).

?WALKBACK
normal return

Input

AC0	Reserved (Set to 0.)
AC1	Reserved (Set to 0.)
AC2	One of the following: • 0 (first ?WALKBACK) • Unchanged (subsequent ?WALKBACKs)
AC3	Frame pointer with carry bit preserved (subsequent ?WALKBACKs only)

Output

AC0	One of the following: • Normal return for this environment if Bit 0 = 0 in AC1 • Procedure descriptor entry for this environment if Bit 0 = 1 in AC1
AC1	Bit 0 is a flag bit: • If Bit 0 = 0, resource call was not used to call this resource • If Bit 0 = 1, resource call was used to call this resource
AC2	One of the following: • Frame pointer for this stack level • 0 (which indicates that no stack frames are left)
AC3	Current frame pointer

Error Codes in AC0

No error codes are currently defined.

Why Use It?

?WALKBACK gives you a picture of the previous stack environment; that is, the contents of the stack after a previous ?KCALL, ?RCALL, or ?RCHAIN sequence. You can issue ?WALKBACK repeatedly, to obtain a complete picture of the stack, or until you reach the desired stack environment.

Who Can Use It?

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

?WALKBACK Continued

What It Does

?WALKBACK traces through the user stack and returns the stack frame pointer, procedure entry descriptor, and return address of the previous resource (the previous stack environment).

The first ?WALKBACK you issue retrieves the most recent stack environment. Before you issue the first ?WALKBACK, set AC2 to 0. On output, AC2 contains the frame pointer for the previous stack environment. Because the operating system uses the contents of the accumulators as input to subsequent ?WALKBACKs, do not alter the accumulators between ?WALKBACKs.

Notes

- See the descriptions of ?KCALL, ?RCALL, and ?RCHAIN in this chapter.

?WDELAY — Suspends a task for a specified time (32–bit processes only).

?WDELAY
error return
normal return

Input

AC0	Delay interval in milliseconds (a double–precision integer)
AC1	Reserved (Set to 0.)
AC2	Reserved (Set to 0.)

Output

AC0	Modified (used by the OS to maintain the delay)
AC1	Undefined
AC2	Undefined

Error Codes in AC0

No error codes are currently defined.

Why Use It?

Usually, you use ?WDELAY to synchronize tasks or to temporarily suspend a task until some asynchronous event has completed.

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

?WDELAY suspends the calling task until the interval that you specify in AC0 has elapsed. If the delay interval is not a multiple of the real–time clock period, the operating system rounds it to the next higher multiple. For example, if the clock frequency is 10 hertz (one period is 100 milliseconds) and the delay interval that you specify is 120 milliseconds, the actual delay is 200 milliseconds. To obtain the current real–time clock frequency, issue ?GHRZ.

?WHIST
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 • 0 to start a histogram for all active processes	
AC1	One of the following: • –1 if AC0 contains a byte pointer • Any other value, if AC0 contains –1, 0, or a PID	
AC2	Address of the histogram packet	

Output

AC0	Unchanged
AC1	Unchanged
AC2	Unchanged

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?

?WHIST allows you to monitor a range of addresses in the calling process or in another process for a specific interval. It also lets you compare the target's CPU time against the CPU time used by other processes, including the operating system. ?WHIST is a useful way to obtain a global view of CPU activity.

Who Can Use It?

There are no special process privileges needed to issue this call, although your process must be a resident one. There are no restrictions concerning file access.

What It Does

?WHIST starts a histogram for a range of local addresses in a 32–bit process. (?IHIST is the analogous system call for 16–bit processes.) The caller cannot activate more than one histogram at a time.

Before you issue ?WHIST, load AC0 with the PID of the target process or with a byte pointer to its process name. If you specify –1 in AC0, the operating system starts a histogram for the calling process. If you specify 0, the operating system compiles histogram statistics for all active processes. Note that your input to AC1 varies, depending on your input to AC0.

You must also set up a histogram packet of ?HWLTH words in your address space and reserve a receive buffer for the histogram statistics. Load the packet address into AC2 before you issue ?WHIST. Figure 2–239 shows the structure of the ?WHIST packet.

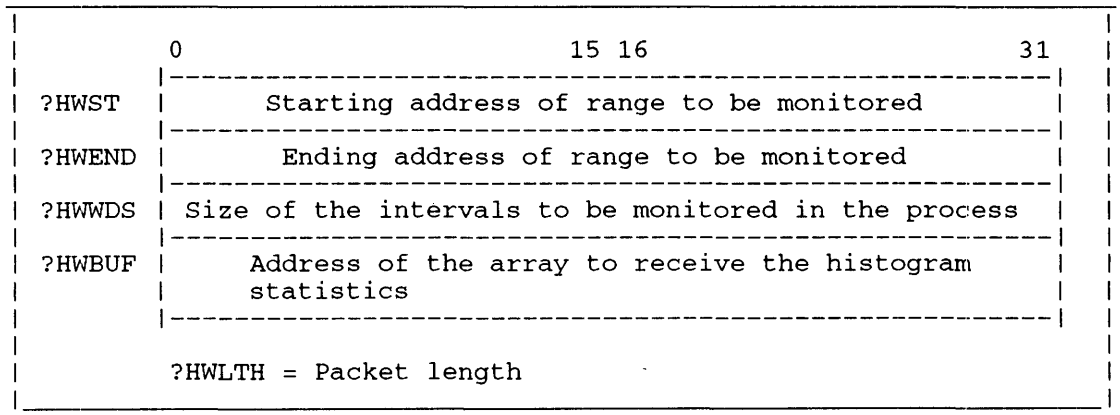


Figure 2–239. Structure of ?WHIST Packet

?HWWDS range is 0 or 1 to *n*, where *n* is the number of words in the range to be monitored. Set ?HWWDS to 0 for a simple histogram, which records how often the target process gained CPU control. The operating system ignores the contents of ?HWST and ?HWEND in a simple histogram.

Figure 2–240 illustrates the use of these parameters. The histogram monitors the entire 64 kbyte context of the target process, and compiles statistics for 256–word intervals through the context.

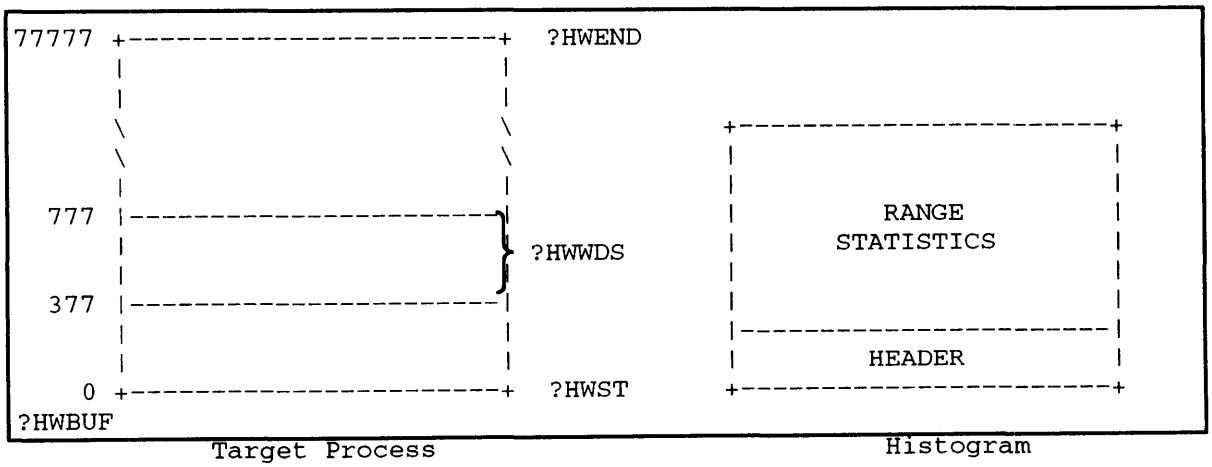


Figure 2–240. Sample Histogram Parameters

The operating system returns the histogram statistics in array format to your receive buffer. The array consists of a fixed–length header, followed by double–precision elements that correspond to each interval you specify. Table 2–210 shows the array structure.

Table 2–210. Histogram Array Structure

Array Offset	Contents
?HTTH ?HTTL	Total number of real-time clock pulses (ticks) counted in this histogram.
?HPRH ?HPRL	Total number of ticks when 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	Total number of ticks in the nth interval.
?HARAY + n * 2 - 1	

Sample Packet

The following sample packet for a simple histogram shows how often the target process gained CPU control:

```
PKT:    .BLK    ?HWLTH          ;Allocate enough space for packet.
                                ;Packet length is ?HWLTH.
        .LOC    PKT+?HWST       ;Starting address of range to be
        .DWORD  0               ;monitored (ignored in simple
                                ;histogram).
        .LOC    PKT+?HWEND      ;Ending address of range to be
        .DWORD  0               ;monitored (ignored in simple
                                ;histogram).
        .LOC    PKT+?HWWDS      ;Size of interval to be monitored.
        .DWORD  0               ;Records how often process gained CPU
                                ;control.
        .LOC    PKT+?HWBUF      ;Address of array to receive
        .DWORD  0               ;histogram statistics (ignored in
                                ;simple histogram).
        .LOC    PKT+?HWLTH      ;End of packet.
```

AOS/VS

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

Input

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

Output

AC0	Unchanged
AC1	Unchanged
AC2	Address of the ?WINDOW packet

Error Codes Returned in AC0

ERBTS	Buffer too small
ERIVS	Invalid view or scan port
ERIWO	Invalid windowing operation
ERIWP	Invalid windowing 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
ERWMD	Window marked for deletion
ERWNE	The window you specified does not exist
ERWNN	Maximum number of windows exceeded

Error codes from the file system

Why Use It?

?WINDOW lets you create and manipulate windows on a physical terminal screen. You can also use ?WINDOW to control keyboard input to windows and to get information about windows.

?WINDOW is useful only if your program will run in an AOS/VS windowing environment. The call applies to a single window (or window group) at a time. For example, to find out the status of every window belonging to your program, you must issue the ?WINDOW 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

?WINDOW is a multifunctional system call. To perform a particular function, you set the offset ?WIN_PKT.FUNC (in the main ?WINDOW packet) to the function code you want. Table 2–211 lists the valid function codes and the functions they perform.

NOTE: Never use the system control processor in a window (unless you are debugging your own operating system).

Table 2–211. ?WINDOW Function Codes

Function Code	What It Lets You Do
?WIN_CREATE_WINDOW	Create a window on the physical terminal screen. (See the section "Creating a New Window.")
?WIN_DELETE_WINDOW	Delete an existing window. (See the section "Deleting a Window.")
?WIN_DEFINE_PORTS	Move or resize the view port or scan port of a window. This effectively lets you move the window on the screen, scroll data in the window, or change the size of the window. (See the section "Changing the View and Scan Ports.")
?WIN_UPFRONT_WINDOW	Make a window the highest priority in its group. (See the section "Controlling Window Priority.")
?WIN_OUTBACK_WINDOW	Make a window the lowest priority in its group. (See the section "Controlling Window Priority.")
?WIN_UPFRONT_GROUP	Make a group of windows the highest priority among groups on the physical terminal. (See the section "Controlling Window Priority.")
?WIN_OUTBACK_GROUP	Make a group of windows the lowest priority among groups on the physical terminal. (See the section "Controlling Window Priority.")
?WIN_HIDE_WINDOW	Make a window invisible to the user. (See the section "Controlling Window Visibility.")
?WIN_UNHIDE_WINDOW	Make a hidden window visible again. (See the section "Controlling Window Visibility.")
?WIN_HIDE_GROUP	Make a group of windows invisible to the user. (See the section "Controlling Window Visibility.")
?WIN_UNHIDE_GROUP	Make a hidden group of windows visible again. (See the section "Controlling Window Visibility.")
?WIN_SUSPEND_WINDOW	Suspend output to a window. (See the section "Controlling Output to a Window or Group.")
?WIN_UNSUSPEND_WINDOW	Resume output to a suspended window. (See the section "Controlling Output to a Window or Group.")
?WIN_SUSPEND_GROUP	Suspend output to a group of windows. (See the section "Controlling Output to a Window or Group.")

(continued)

Table 2–211. ?WINDOW Function Codes

Function Code	What It Lets You Do
?WIN_UNSUSPEND_GROUP	Resume output to a suspended group of windows. (See "Controlling Output to a Window or Group.")
?WIN_ENABLE_KEYBOARD	Allow keyboard input to a window's input buffer. (See the section "Controlling Keyboard Input.")
?WIN_DISABLE_KEYBOARD	Prevent keyboard input from going to a window. (See the section "Controlling Keyboard Input.")
?WIN_PERMANENCE_ON	Set PERMANENCE on for a window. (See the section "Setting a Window's Permanence.")
?WIN_PERMANENCE_OFF	Set PERMANENCE off for a window. (See the section "Setting a Window's Permanence.") Permanence.")
?WIN_SET_USER_INTERFACE	Determine the appearance of the window border; specify whether or not the user can move, resize or scroll the window. (See the section "Setting the User Interface.")
?WIN_GET_USER_INTERFACE	Find out the current user interface settings for a window. (See the section "Getting the Current User Interface Settings.")
?WIN_SET_TITLE	Set the title of a window. (See the section "Setting a Window's Title.")
?WIN_GET_TITLE	Get the title of a window. (See the section "Getting a Window's Title.")
?WIN_GET_WINDOW_NAME	Get the pathname of a window. (See the section "Getting a Window's Pathname.")
?WIN_WINDOW_STATUS	Get the current status of a window. (See the section "Getting a Window's Status.")
?WIN_DEVICE_STATUS	Get the current status of a physical device. (See the section "Getting the Status of a Physical Device.")
?WIN_GET_WINDOW_ID	Get the window ID of a window. (See the section "Getting Window IDs.")
?WIN_RETURN_ GROUP_WINDOWS	Get the window IDs of all windows in a particular group. (See the section "Getting Window IDs.")
?WIN_RETURN_ DEVICE_WINDOWS	Get the window IDs of all windows on the physical terminal. (See the section "Getting Window IDs.")

(concluded)

The Main ?WINDOW Packet

When you issue a ?WINDOW call, you usually set up both a main packet (common to all functions of the call) and a subpacket (unique for each function). Most functions require subpackets. The structure for the main packet appears in Figure 2–241; Table 2–212 details its contents. We cover the subpackets later, when we discuss each function.

The ?WINDOW call usually applies to one window at a time, or to a particular window group. In the main packet, you specify which window or group you want. You can specify a window in any 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 the 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 packet.

To specify a group of windows, simply specify a window within that group. Set the offset ?WIN_PKT.FLAGS to indicate which method you are using to specify the window or group. 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.

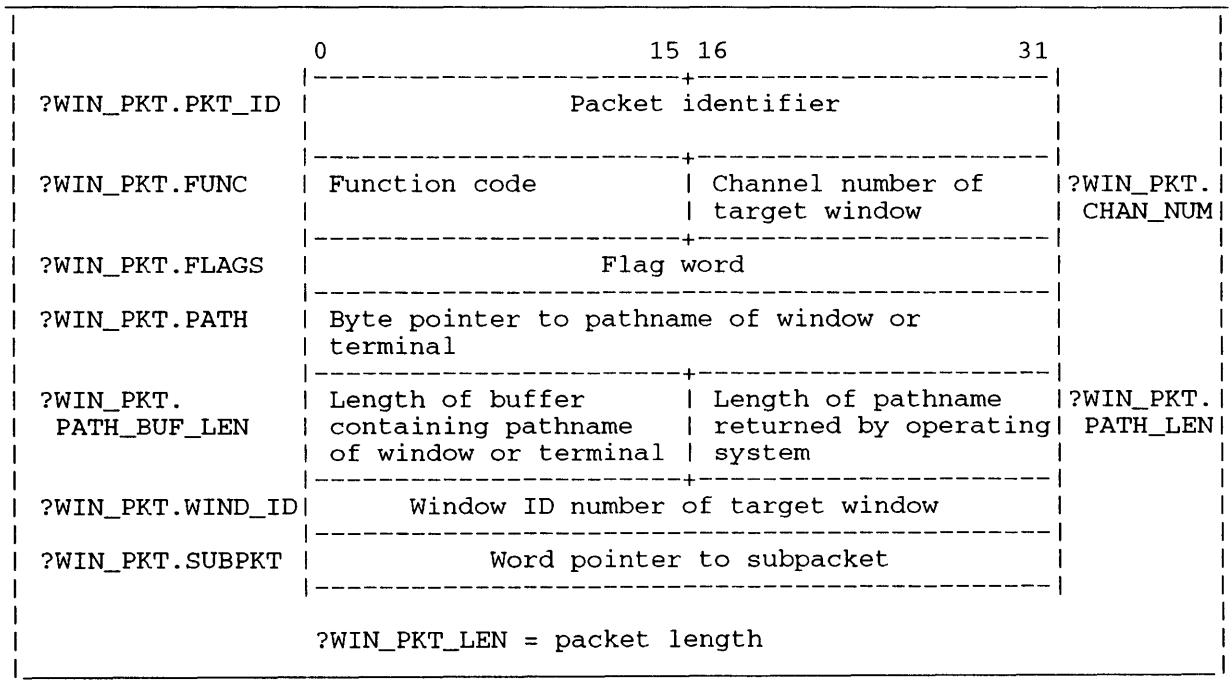


Figure 2–241. Structure of the ?WINDOW Main Packet

Table 2-212. Contents of the ?WINDOW Main Packet

Offset	Contents
?WIN_PKT.PKT_ID (doubleword)	Packet identifier; set to ?WIN_PKT_PKTID.
?WIN_PKT.FUNC	Code for the function you want. You must always supply a function code. (Function codes are listed in Table 2-211.)
?WIN_PKT.CHAN_NUM	Channel number of target window. When specifying the window by its channel number, you must also set flag ?WIN_PKT.FLAGS.IN_CHAN in flag word ?WIN_PKT.FLAGS. Set this offset to 0 when specifying the target window by its pathname or window ID.
?WIN_PKT.FLAGS (doubleword)	Flag word. ?WIN_PKT.FLAGS.IN_PATH -- Set this flag when specifying the target window by its pathname. ?WIN_PKT.FLAGS.IN_WIND_ID -- Set this flag when specifying target window by its window ID. ?WIN_PKT.FLAGS.IN_CHAN -- Set this flag when specifying target window by its channel number.
?WIN_PKT.PATH (doubleword)	Byte pointer to pathname of target window. When specifying the window by its pathname, you must also set flag ?WIN_PKT.FLAGS.IN_PATH in flag word ?WIN_PKT.FLAGS. Set this offset to 0 when specifying the target window by its channel number or window ID.
?WIN_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.
?WIN_PKT.PATH_LEN	Unused. (Set to 0.)
?WIN_PKT.WIND_ID (doubleword)	Window ID number of target window. When specifying the window by its window ID, you must also set flag ?WIN_PKT.FLAGS.IN_WIND_ID in flag word ?WIN_PKT.FLAGS. Set this offset to 0 when specifying the target window by its pathname or channel number.
?WIN_PKT.SUBPKT	Word pointer to the subpacket for the function. If the function does not require a subpacket, set this offset to 0.

Creating a New Window

To create a new window on a physical terminal, issue the ?WINDOW call, function code ?WIN_CREATE_WINDOW. You can create a new window only if you have access to the physical terminal, or to an existing window on that terminal.

If you have access to the physical terminal, you can specify the terminal's device pathname in the main packet. The new window will then become the first window in a new window group. (If you specify a device pathname in the main packet, you must also set the flag ?WIN_CRE.FLAGS.CREATE_SON in the subpacket flag word ?WIN_CRE.FLAGS.)

If you have access to an existing window on the terminal, you can specify that window's pathname, channel number, or window ID in the main packet. The new window can either begin a new group or join the existing window's group.

In the ?WIN_CREATE_WINDOW subpacket, you must indicate whether you specified a window or a physical terminal in the main packet, and whether or not the new window will begin a new group. You can also specify some of the new window's attributes in the subpacket. Figure 2–242 shows the subpacket structure; Table 2–213 details its contents.

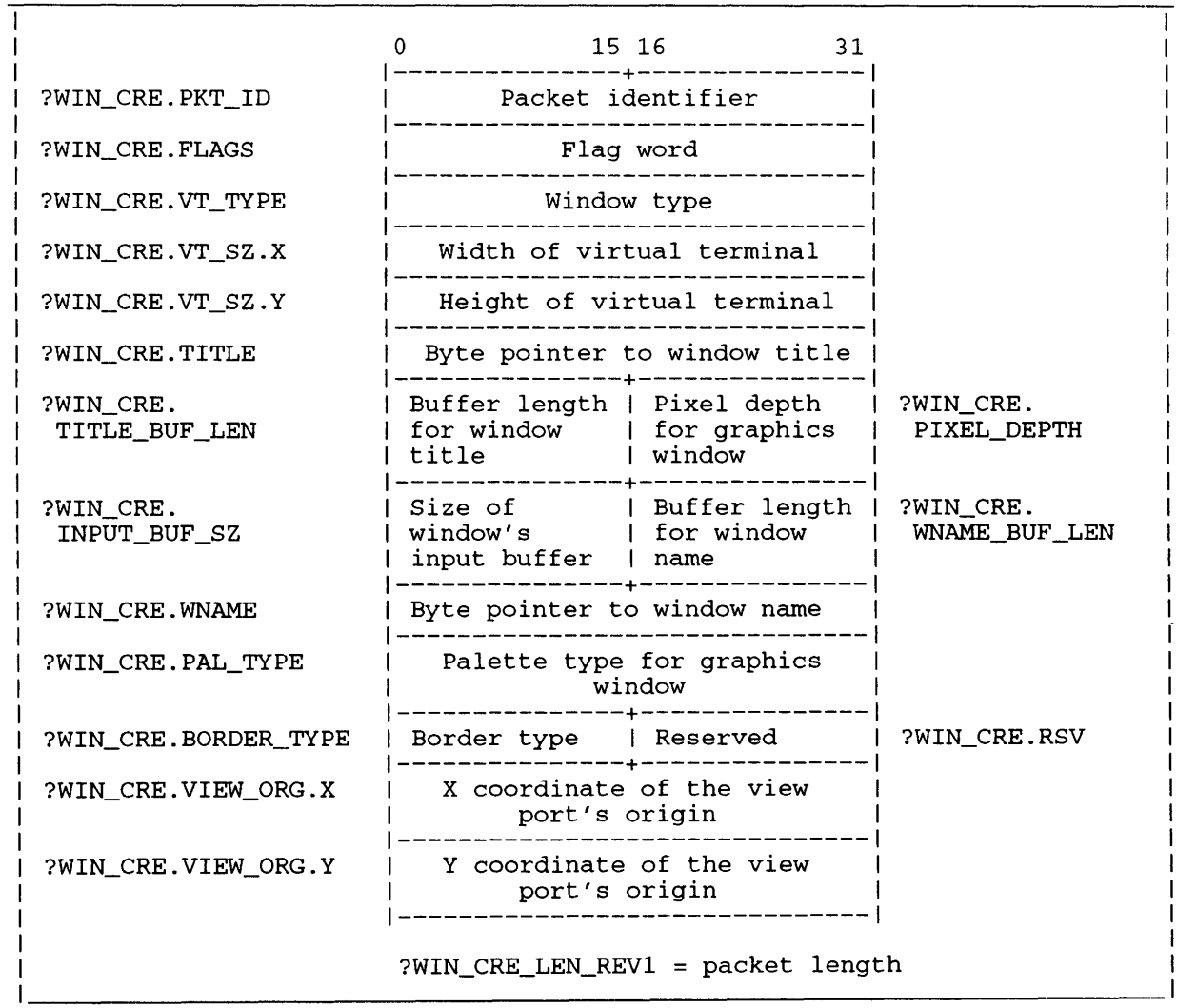


Figure 2–242. Structure of the ?WIN_CREATE_WINDOW Subpacket

Table 2–213. Contents of the ?WIN_CREATE_WINDOW Subpacket

Offset	Contents
?WIN_CRE.PKT_ID (doubleword)	Packet identifier; set to ?WIN_CRE_PKTID_REV1.
?WIN_CRE.FLAGS (doubleword)	Flag word. ?WIN_CRE.FLAGS.NEW_GRP -- Set this flag if the new window will be the first window in a new group. You must set this flag if you specify a physical terminal's pathname in the main packet. ?WIN_CRE.FLAGS.VT_SZ_X -- Set this flag if you are specifying the width of the virtual terminal; set offset ?WIN_CRE.VT_SZ_X to the value you want. If you do not set this flag, the operating system uses the default width. ?WIN_CRE.FLAGS.VT_SZ_Y -- Set this flag if you are specifying the width of the virtual terminal; set offset ?WIN_CRE.VT_SZ_Y to the value you want. If you do not set this flag, the operating system uses the default height. ?WIN_CRE.FLAGS.PIXEL_DEPTH -- Set this flag if you are specifying the number of bits per pixel for a graphics window; set offset ?WIN_CRE.PIXEL_DEPTH to the value you want. If you do not set this flag, the operating system uses the default pixel depth. ?WIN_CRE.FLAGS.CREATE_SON -- You must set this flag if you specify a physical terminal's pathname in the main packet (the new window will be a "son window" of the terminal). If you set this flag, you must also set the flag ?WIN_CRE.FLAGS.NEW_GRP. ?WIN_CRE.FLAGS.VIEW_ORG.X -- You must set this flag if you are specifying an X coordinate for the view port's origin; set offset ?WIN_CRE.VIEW_ORG.X to the value you want. If you do not set this flag, the operating system uses a default value. ?WIN_CRE.FLAGS.VIEW_ORG.Y -- You must set this flag if you are specifying a Y coordinate for the view port's origin; set offset ?WIN_CRE.VIEW_ORG.Y to the value you want. If you do not set this flag, the operating system uses a default value. ?WIN_CRE.FLAGS.UNHIDE_WINDOW -- Set this flag to make the window visible immediately after creation. If you do not set this flag, the window remains hidden until you use the ?WINDOW function ?WIN_UNHIDE_WINDOW to make it visible.

(continued)

Table 2–213. Contents of the ?WIN_CREATE_WINDOW Subpacket

Offset	Contents
?WIN_CRE.FLAGS (doubleword) (continued)	?WIN_CRE.FLAGS.INPUT_BUF_SZ -- Set this flag if you are specifying the size of the window's input buffer; set offset ?WIN_CRE.INPUT_BUF_SZ to the value you want. If you do not set this flag, the operating system uses the size of the input buffer associated with the physical terminal.
?WIN_CRE.VT_TYPE (doubleword)	Type of virtual terminal the new window will have. Select from the following values. ?WIN_VT_TYPE_DEFAULT — The operating system determines the virtual terminal type. ?WIN_VT_TYPE_D460 — (Character window with 2 bits per pixel.) The virtual terminal will be a software emulation of a D460 terminal and will support the blink attribute. ?WIN_VT_TYPE_PIXMAP — (Graphics window.) The virtual terminal will be a pixel map.
?WIN_CRE.VT_SZ.X (doubleword)	Width of the new window's virtual terminal. For a character window (?WIN_VT_TYPE_D460),specify the width in characters. For a graphics window (?WIN_VT_TYPE_PIXMAP), specify the width in pixels. When specifying virtual terminal width, you must set the flag ?WIN_CRE.FLAGS.VT_SZ_X in flag word ?WIN_CRE.FLAGS.
?WIN_CRE.VT_SZ.Y (doubleword)	Height of the new window's virtual terminal. For a character window (?WIN_VT_TYPE_D460), specify the height in characters. For a graphics window (?WIN_VT_TYPE_PIXMAP), specify the height in pixels. When specifying virtual terminal height, you must set the flag ?WIN_CRE.FLAGS.VT_SZ_Y in flag word ?WIN_CRE.FLAGS.

(continued)

Table 2–213. Contents of the ?WIN_CREATE_WINDOW Subpacket

Offset	Contents
?WIN_CRE.TITLE (doubleword)	Byte pointer to the buffer containing the new window's title. The window title is a character string up to 32 characters long; the last character must be a null. You must always provide a byte pointer. To create a window with no title, provide a byte pointer to a null title string.
?WIN_CRE. TITLE_BUF_LEN	The length of the buffer containing the window title. The maximum length for the window title is ?MXFN.
?WIN_CRE. PIXEL_DEPTH	The number of bits per pixel, for a graphics window.
?WIN_CRE. INPUT_BUF_SZ	The length of the input buffer associated with the new window's virtual terminal. The buffer length must be at least 32. When specifying input buffer length, you must also set the flag ?WIN_CRE.FLAGS.INPUT_BUF_SZ in flag word ?WIN_CRE.FLAGS.
?WIN_CRE. WNAME_BUF_LEN	The length of the buffer containing the window name. The maximum length for the window name is ?MXFN.
?WIN_CRE.WNAME (doubleword)	Byte pointer to the buffer containing the new window's window name. The window name is a character string up to ?MXFN characters long. You must always provide a window name, since it forms part of the window's pathname.
?WIN_CRE.PAL_TYPE (doubleword)	Type of palette the new window will have. Select from the following values. ?WIN_PALETTE_TYPE_DEFAULT -- The system determines the palette type for this window. ?WIN_PALETTE_TYPE_USER -- User-definable palette. It has 2**(pixel depth) entries. For character windows, the pixel depth depends on the virtual terminal type. For graphics windows, you specify the pixel depth in offset ?WIN_CRE.PIXEL_DEPTH. All entries in the palette are initially set to "dark".

(continued)

Table 2–213. Contents of the ?WIN_CREATE_WINDOW Subpacket

Offset	Contents
?WIN_CRE.PAL_TYPE (doubleword) (continued)	?WIN_PALETTE_TYPE_SYSTEM -- System palette. The window shares the system palette, which is two entries long. The window cannot write to this palette. ?WIN_PALETTE_TYPE_SHARED -- Shared palette. The window shares the palette of the existing window specified in the main ?WINDOW packet.
?WIN_CRE. BORDER_TYPE	Type of border the new window will have. Select from the following values. ?WIN_BORDER_TYPE_DEFAULT — System determined border type. ?WIN_BORDER_TYPE_NONE — No border. ?WIN_BORDER_TYPE_LINE — Simple line border. ?WIN_BORDER_TYPE_TITLE — Line border with space at the top to display a window title. ?WIN_BORDER_TYPE_INTELLIGENT — Intelligent border.
?WIN_CRE. VIEW_ORG.X (doubleword)	The initial X coordinate of the view port's origin (in pixels relative to the origin of the physical screen). To use the system default value, set to 0.
?WIN_CRE. VIEW_ORG.Y (doubleword)	The initial Y coordinate of the view port's origin (in pixels relative to the origin of the physical screen). To use the system default value, set to 0.

(concluded)

Deleting a Window

To delete an existing window, issue ?WINDOW with function code ?WIN_DELETE_WINDOW. If the window is in use (that is, if a channel is open to it, or it is still assigned to a process) the operating system will not delete it until the last process deassigns it. In the main packet, provide a pathname, channel number or window ID for the window you want to delete. There is no subpacket for ?WIN_DELETE_WINDOW; set offset ?WIN_PKT.SUBPKT to 0.

Changing the View and Scan Ports

Every window has a view port (the portion of the physical screen within the window border) and a scan port (the portion of the virtual screen that's within the view port). The view port is always the same size as the scan port.

You can alter the window's size and position by changing the size and location of its ports. To do so, issue the ?WINDOW call with function code ?WIN_DEFINE_PORTS; in the main packet, provide a pathname, channel number or window ID for the window you want to change.

- To move the window on the physical screen, alter the origin of its view port.
- To scroll data in the window (move the scan port on the virtual screen), alter the origin of its scan port.
- To change the size of the window, alter the size of its scan port; the operating system adjusts the size of the view port to match.

You specify coordinate values in the ?WIN_DEFINE_PORTS subpacket. Figure 2–243 shows the subpacket structure; Table 2–214 details its contents.

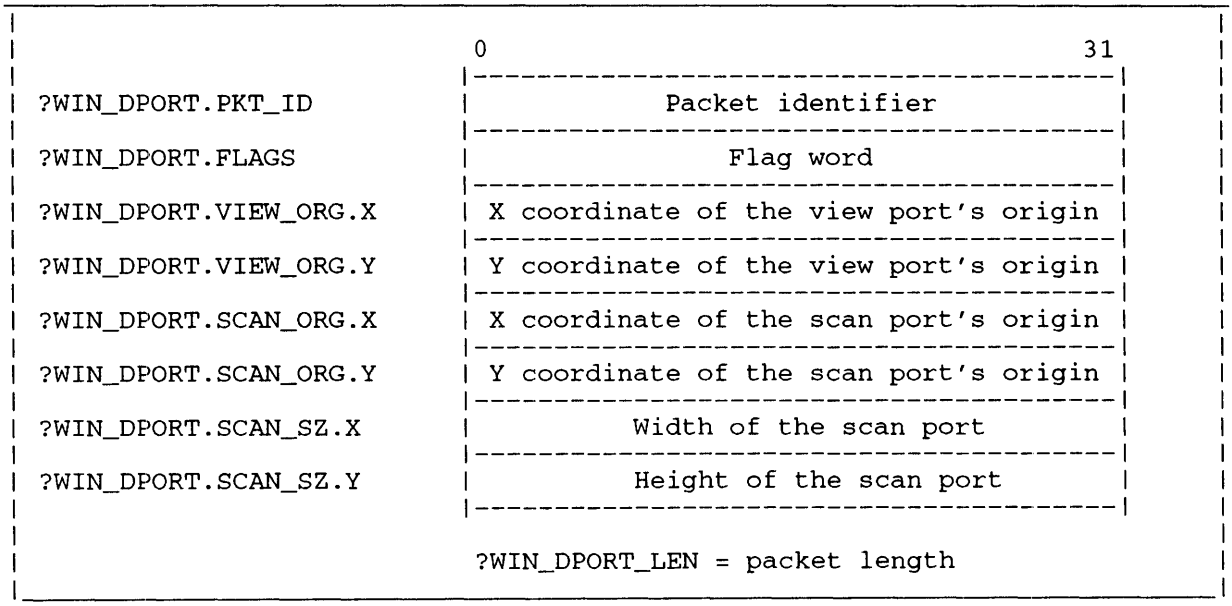


Figure 2–243. Structure of the ?WIN_DEFINE_PORTS Subpacket

For each view or scan port parameter you want to change, you must set the corresponding flag in the flag word ?WIN_DPORT.FLAGS. If you are not changing a particular parameter, set both the flag and the parameter offset to 0.

Table 2–214. Contents of the ?WIN_DEFINE_PORTS Subpacket

Offset	Contents
?WIN_DPORT.PKT_ID (doubleword)	Packet identifier; set to ?WIN_DPORT_PKTID.
?WIN_DPORT.FLAGS (doubleword)	Flag word. For each value you change, you must set the corresponding flag, as follows. ?WIN_DPORT.FLAGS.VIEW_ORG_X -- Set this flag if you are changing the X coordinate of view port's origin. ?WIN_DPORT.FLAGS.VIEW_ORG_Y -- Set this flag if you are changing the Y coordinate of view port's origin. ?WIN_DPORT.FLAGS.SCAN_ORG_X -- Set this flag if you are changing the X coordinate of scan port's origin. ?WIN_DPORT.FLAGS.SCAN_ORG_Y -- Set this flag if you are changing the Y coordinate of scan port's origin. ?WIN_DPORT.FLAGS.SCAN_SZ_X -- Set this flag if you are changing the width of the scan port. ?WIN_DPORT.FLAGS.SCAN_SZ_Y -- Set this flag if you are changing the height of the scan port.
?WIN_DPORT.VIEW_ORG.X (doubleword)	X coordinate of the view port's origin (in pixels relative to the origin of the physical screen). Set to 0 if you are not changing this value.
?WIN_DPORT.VIEW_ORG.Y (doubleword)	Y coordinate of the view port's origin (in pixels relative to the origin of the physical screen). Set to 0 if you are not changing this value.
?WIN_DPORT.SCAN_ORG.X (doubleword)	X coordinate of the scan port's origin (in pixels for a graphics window, characters for a character window); relative to the origin of the virtual screen. Set to 0 if you are not changing this value.
?WIN_DPORT.SCAN_ORG.Y (doubleword)	Y coordinate of the scan port's origin (in pixels for a graphics window, characters for a character window); relative to the origin of the virtual screen. Set to 0 if you are not changing this value.
?WIN_DPORT.SCAN_SZ.X (doubleword)	Width of the scan port (in pixels for a graphics window, characters for a character window). Set to 0 if you are not changing this value.
?WIN_DPORT.SCAN_SZ.Y (doubleword)	Height of the scan port (in pixels for a graphics window, character for a character window). Set to 0 if you are not changing th value.

Controlling Window Priority

The operating system assigns a unique window priority to each window on the screen. The window priority determines the front–to–back order of the windows: where windows overlap, the window with the higher priority appears in front of any window(s) with lower priorities. Window groups are collections of windows that the operating system maintains at consecutive priorities. Each group can be considered to have a priority relative to other groups.

You use the ?WINDOW call to change the priority of a window within its group, or to change the priority of a window group relative to other groups on the screen.

To Bring a Window or Group to the Front

The ?WIN_UPFRONT_WINDOW function makes a window the highest priority among windows in its group; when visible, that window will then appear in front of other windows in its group. (This function does not alter the window's visibility.) In the main packet, provide a pathname, channel number or window ID for the window you want to bring to the front. There is no subpacket for ?WIN_UPFRONT_WINDOW; set offset ?WIN_PKT.SUBPKT to 0.

The ?WIN_UPFRONT_GROUP function makes a window group the ghest priority among groups on the screen. When that group is visible, it will be the active group, and its windows will appear in front of windows belonging to other groups. (This function does not alter the group's visibility or the internal hierarchy of its windows.) In the main packet, provide a pathname, channel number or window ID for any window in the group you want to bring to the front. There is no subpacket for ?WIN_UPFRONT_GROUP; set offset ?WIN_PKT.SUBPKT to 0.

To Send a Window or Group to the Back

The ?WIN_OUTBACK_WINDOW function makes a window the lowest priority among windows in its group; when visible, that window will then appear behind other windows in its group. In the main packet, provide a pathname, channel number or window ID for the window you want to send to the back. There is no subpacket for ?WIN_OUTBACK_WINDOW; set offset ?WIN_PKT.SUBPKT to 0.

The ?WIN_OUTBACK_GROUP function makes a group of windows the lowest priority among groups on the screen; when that group is visible, its windows will then appear behind windows in other groups. If you issue this function for the active group, the operating system makes the visible group with the highest priority the new active group. In the main packet, provide a pathname, channel number or window ID for any window in the group you want to send to the back. There is no subpacket for ?WIN_OUTBACK_GROUP; set offset ?WIN_PKT.SUBPKT to 0.

Controlling Window Visibility

A window on the screen can be either visible or invisible to the user. The window's visibility does not affect its priority (its place within the hierarchy of windows in its group). Likewise, hiding a group of windows does not affect that group's priority relative to other groups, or its internal hierarchy of windows.

To Make a Window or Group Invisible

The ?WIN_HIDE_WINDOW function makes a window invisible to the user. Although a hidden window does not appear on the screen, it keeps its place in the hierarchy of windows within its group.

In the main packet, provide a pathname, channel number or window ID for the window you want to hide. There is no subpacket for ?WIN_HIDE_WINDOW; set offset ?WIN_PKT.SUBPKT to 0.

The ?WIN_HIDE_GROUP function makes a group of windows invisible to the user. This function does not affect the group's place within the hierarchy of groups on the screen, or the hierarchy of windows within the group. If you hide the active group, the visible group with the highest priority becomes the new active group.

In the main packet, provide a pathname, channel number or window ID for any window in the group you want to hide. There is no subpacket for ?WIN_HIDE_GROUP; set offset ?WIN_PKT.SUBPKT to 0.

To Make a Hidden Window or Group Visible

The ?WIN_UNHIDE_WINDOW function makes a hidden window visible again. In the main packet, provide a pathname, channel number or window ID for the window you want to make visible. There is no subpacket for ?WIN_UNHIDE_WINDOW; set offset ?WIN_PKT.SUBPKT to 0.

The ?WIN_UNHIDE_GROUP function makes a hidden group of windows visible to the user. (This function does not affect the group's priority relative to other groups, or the internal hierarchy of windows within that group. It also does not affect the visibility of individual windows: if a window in the group was previously hidden individually, it remains hidden even after you make its group visible.)

In the main packet, provide a pathname, channel number or window ID for any window in the group you want to make visible. There is no subpacket for ?WIN_UNHIDE_GROUP; set offset ?WIN_PKT.SUBPKT to 0.

Controlling Output

You can use ?WINDOW to suspend or resume output to a window's virtual screen. When you suspend a window, output to that window's virtual screen stops as if the user had pressed Ctrl–S. You can suspend either a single window or a group of windows. When you resume output to a window or group, output to the virtual screen(s) resumes as if the user had pressed Ctrl–Q.

To Suspend Output

To suspend output to a window, issue the ?WINDOW call with function code ?WIN_SUSPEND_WINDOW. In the main packet, provide a pathname, channel number or window ID for the window you want to suspend. There is no subpacket for ?WIN_SUSPEND_WINDOW; set offset ?WIN_PKT.SUBPKT to 0.

To suspend output to an entire group of windows, issue ?WINDOW with function code ?WIN_SUSPEND_GROUP. In the main packet, provide a pathname, channel number or window ID for any window in the group you want to suspend. There is no subpacket for ?WIN_SUSPEND_GROUP; set offset ?WIN_PKT.SUBPKT to 0.

To Resume Output

To resume output to a suspended window, issue ?WINDOW with function code ?WIN_UNSUSPEND_WINDOW. In the main packet, provide a pathname, channel number or window ID for the window to which you want to resume output. There is no subpacket for ?WIN_UNSUSPEND_WINDOW; set offset ?WIN_PKT.SUBPKT to 0.

To resume output to a suspended group of windows, issue ?WINDOW with function code ?WIN_UNSUSPEND_GROUP. (If a window in the group was previously suspended individually, it remains suspended until you issue a ?WINDOW call, function code ?WIN_UNSUSPEND_WINDOW.) In the main packet, provide a pathname, channel number or window ID for any window in the group to which you want to resume output. There is no subpacket for ?WIN_UNSUSPEND_GROUP; set offset ?WIN_PKT.SUBPKT to 0.

Controlling Keyboard Input

?WINDOW lets you control whether or not keyboard input enters a window's input data stream. By default, keyboard input is enabled.

To prevent keyboard input from entering a window's input data stream, issue ?WINDOW, function code ?WIN_DISABLE_KEYBOARD. In the main packet, provide a pathname, channel number or window ID for the window you want to disable. There is no subpacket for ?WIN_DISABLE_KEYBOARD; set offset ?WIN_PKT.SUBPKT to 0.

To allow keyboard input to go to a window for which you previously disabled keyboard input, issue ?WINDOW with function code ?WIN_ENABLE_KEYBOARD. In the main packet, provide a pathname, channel number or window ID for the window you want to enable. There is no subpacket for ?WIN_ENABLE_KEYBOARD; set offset ?WIN_PKT.SUBPKT to 0.

Setting a Window's Permanence

Windows have a PERMANENCE attribute just as files do. When a window's PERMANENCE is on, you cannot delete that window.

Use function ?WIN_PERMANENCE_ON to turn PERMANENCE on for a window. In the main packet, provide a pathname, channel number or window ID for the window you want to protect. There is no subpacket for ?WIN_PERMANENCE_ON; set offset ?WIN_PKT.SUBPKT to 0.

Use function ?WIN_PERMANENCE_OFF to turn window PERMANENCE off. In the main packet, provide a pathname, channel number or window ID for the window you want. There is no subpacket for ?WIN_PERMANENCE_OFF; set offset ?WIN_PKT.SUBPKT to 0.

Setting the User Interface

You can control some aspects of a window's appearance using the ?WINDOW call with function code ?WIN_SET_USER_INTERFACE. This function lets you determine

- The type of window border (intelligent, title, line, or none).
- Whether or not the user can move the window.
- Whether or not the user can change the window's size.
- Whether or not the user can scroll data in the window.
- Whether or not to maintaielevators on an intelligent border.
- Whether or not the user can request help by selecting the help symbol on an intelligent border.
- Whether or not the user can make a window full sized and return it to its original size.
- Whether or not the user can move the borders off the screen.

NOTE: The user alters windows using DG/VIEW, a menu–driven windowing interface. If DG/VIEW is not available, the user will not be able to change the window, regardless of the settings in the ?WIN_SET_USER_INTERFACE subpacket.

In the main packet, specify the window ID, pathname or channel number of the window you want to change. You specify the changes you want in the subpacket. Figure 2–244 shows the structure of the subpacket; Tables 2–215 and 2–216 detail its contents.

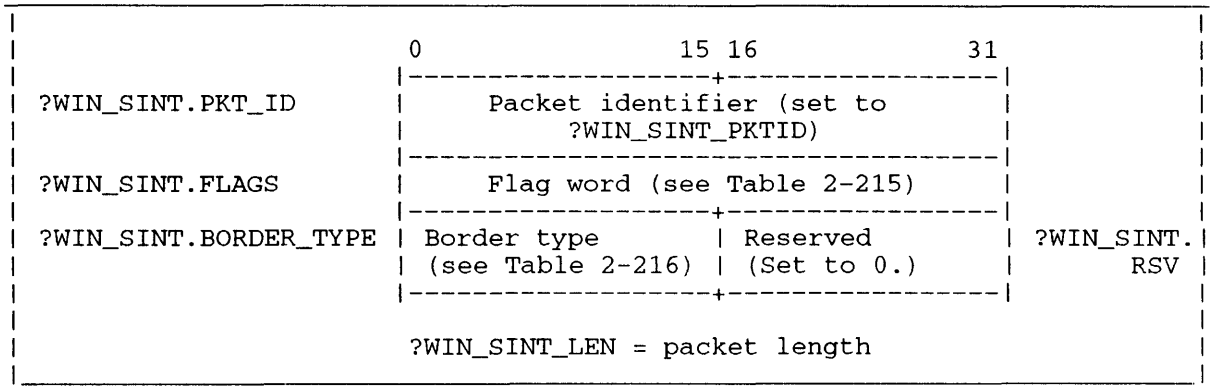


Figure 2–244. Structure of the ?WIN_SET_USER_INTERFACE Subpacket

Table 2–215. Flags in Flag Word ?WIN_SINT.FLAGS

Flag	Description
?WIN_SINT.FLAGS.MOVE_X	Set this flag to let the user move the window along the x-axis.
?WIN_SINT.FLAGS.MOVE_Y	Set this flag to let the user move the window along the y-axis.
?WIN_SINT.FLAGS.SZ_X	Set this flag to let the user change the width of the window.
?WIN_SINT.FLAGS.SZ_Y	Set this flag to let the user change the height of the window.
?WIN_SINT.FLAGS.SCROLL_X	Set this flag to let the user scroll data horizontally in the window.
?WIN_SINT.FLAGS.SCROLL_Y	Set this flag to let the user scroll data vertically in the window.
?WIN_SINT.FLAGS.ELEVATORS	Set this flag to maintain elevators on the intelligent border.
?WIN_SINT.FLAGS.FULL_REV	Set this flag to let the user expand the window to full size and revert it back to its original size.
?WIN_SINT.FLAGS.HELP	Set this flag to let the user select the help symbol on the intelligent border.
?WIN_SINT.FLAGS.WIRE_BRDRS	Set this flag to prevent the user from moving window borders off the screen.

Table 2–216. Border Types (Offset ?WIN_SINT.BORDER_TYPE)

Value	Description
?WIN_BORDER_TYPE_DEFAULT	System determined border type.
?WIN_BORDER_TYPE_NONE	No border.
?WIN_BORDER_TYPE_LINE	Simple line border.
?WIN_BORDER_TYPE_TITLE	Line border with space at the top to display a window title.
?WIN_BORDER_TYPE_INTELLIGENT	Intelligent border.

Getting the Current User Interface Settings

To find out a window's current user interface settings, issue the ?WINDOW call with function code ?WIN_GET_USER_INTERFACE. In the main packet, specify the window ID, pathname or channel number of the window whose user interface settings you want. The operating system returns the information in the subpacket. Figure 2–245 shows the structure of the subpacket; Tables 2–217 and 2–218 detail its contents.

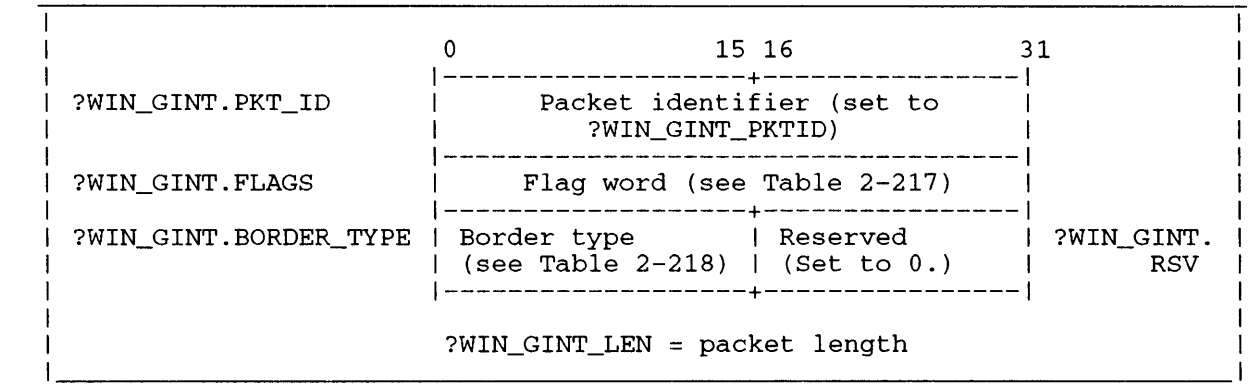


Figure 2-245. Structure of the ?WIN_GET_USER_INTERFACE Subpacket

NOTE: The user alters windows using DG/VIEW, a menu–driven windowing interface. If DG/VIEW is not available, the user will not be able to change the window, regardless of the settings returned in the ?WIN_GET_USER_INTERFACE subpacket.

Table 2–217. Flags in the ?WIN_GET_USER_INTERFACE Flag Word (Flag Word ?WIN_GINT.FLAGS)

Flag	Description
?WIN_GINT.FLAGS.MOVE_X	This flag is set if the user can move the window horizontally.
?WIN_GINT.FLAGS.MOVE_Y	This flag is set if the user can move the window vertically.
?WIN_GINT.FLAGS.SZ_X	This flag is set if the user can change the width of the window.
?WIN_GINT.FLAGS.SZ_Y	This flag is set if the user can change the height of the window.
?WIN_GINT.FLAGS.SCROLL_X	This flag is set if the user can scroll data horizontally in the window.
?WIN_GINT.FLAGS.SCROLL_Y	This flag is set if the user can scroll data vertically in the window.
?WIN_GINT.FLAGS.ELEVATORS	This flag is set if elevators are maintained on the intelligent border.
?WIN_GINT.FLAGS.FULL_REV	This flag is set if the user can expand the window to full size and revert it back to its original size.
?WIN_GINT.FLAGS.HELP	This flag is set if the user can select the help symbol on the intelligent border.
?WIN_GINT.FLAGS.WIRE_BRDRS	This flag is set if the user cannot move window borders off the screen.

Table 2–218. Border Types (Offset ?WIN_GINT.BORDER_TYPE)

Value	Description
?WIN_BORDER_TYPE_DEFAULT	System determined border type.
?WIN_BORDER_TYPE_NONE	No border.
?WIN_BORDER_TYPE_LINE	Simple line border.
?WIN_BORDER_TYPE_TITLE	Line border with space at the top to display a window title.
?WIN_BORDER_TYPE_INTELLIGENT	Intelligent border.

Setting a Window's Title

If a window has an intelligent or a title border, a window title can appear within the border. The window title only identifies the window to the user; you cannot refer to a window by its title when issuing a system call.

To set the window title, issue ?WINDOW with function code ?WIN_SET_TITLE; in the main packet, provide a pathname, channel number, or window ID for the window whose title you want to set. In the subpacket, specify the size and location of the buffer that contains the title string.

The title string always ends in the null character, and can be up to ?MXFN characters long (including the null terminator). Every window has a title string; the title string for a window with no title is the null string.

Figure 2–246 shows the subpacket structure.

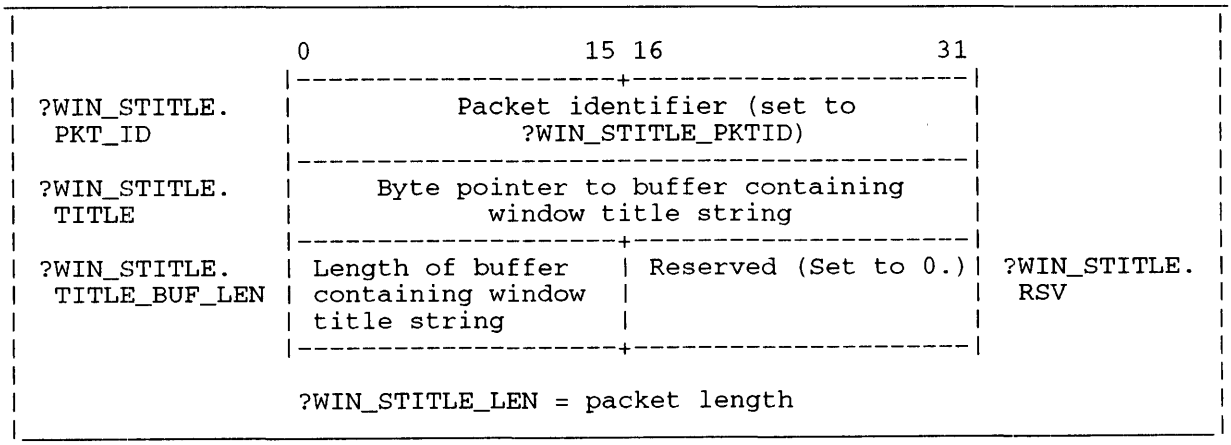


Figure 2–246. Structure of the ?WIN_GET_TITLE Subpacket

Getting a Window's Title

To find out the current title of a window, issue ?WINDOW with function code ?WIN_GET_TITLE; in the main packet, provide a pathname, channel number or window ID for the window whose title you want. In the subpacket, specify the size and location of the buffer that will contain the title string. (A window's title can be up to ?MXFN characters long, including the null terminator. Every window has a title string; the title string for a window with no title is the null string.)

The operating system returns a byte pointer to the window's title string. If the buffer you specified was too small to contain the string, the operating system takes the error return, and returns the length of the title string in offset ?WIN_GTITLE_TITLE_LEN.

Figure 2–247 shows the subpacket structure; Table 2–219 details its contents.

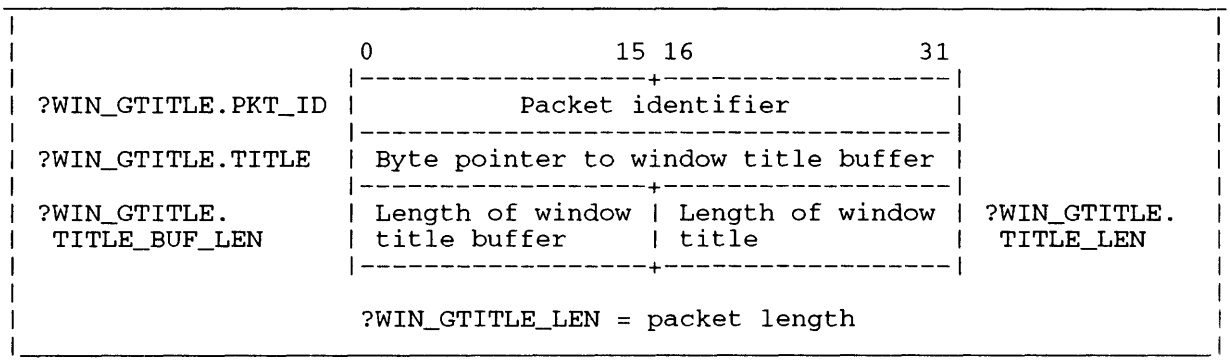


Figure 2–247. Structure of the ?WIN_GTITLE Subpacket

Table 2–219. Contents of the ?WIN_GTITLE Subpacket

Offset	Contents
?WIN_GTITLE.PKT_ID (doubleword)	Packet identifier; set to ?WIN_GTITLE_PKTID.
?WIN_GTITLE.TITLE (doubleword)	Provide a byte pointer to the buffer that will contain the title string returned by the operating system.
?WIN_GTITLE.TITLE_BUF_LEN	Supply the length of the buffer that will contain the title string. The maximum title length is ?MXFN.
?WIN_GTITLE.TITLE_LEN	Set to 0. If the title buffer is too short to hold the title string, the operating system returns the length of the string here (including the null terminator).

To find out a window's pathname, issue ?WINDOW with function code ?WIN_GET_WINDOW_NAME; in the main packet, specify the window ID or channel number of the window you want. There is no subpacket for the ?WIN_GET_WINDOW_NAME function; set ?WIN_PKT.SUBPKT to 0.

The operating system returns a byte pointer to the window's pathname in offset ?WIN_PKT.PATH. It returns the length of the pathname (including the null terminator) in ?WIN_PKT.PATH_LEN.

Getting a Window's Status

The ?WINDOW function ?WIN_WINDOW_STATUS returns the following information about a window:

- Whether or not the window (or its group) is hidden.
- Whether or not the window (or its group) is suspended.
- Whether or not the window's keyboard is enabled.
- Whether or not window PERMANENCE is on.
- The size and location of the window's view port.
- The size and location of the window's scan port.
- The size and location of the window on the screen.
- The window's pixel depth (for graphics windows).
- The size of the window's input buffer.
- The type of palette associated with the window.

In the main packet, specify the window ID, pathname or channel number of the window you want. The operating system returns information in the subpacket. Figure 2–248 shows the structure of the subpacket; Table 2–220 details its contents.

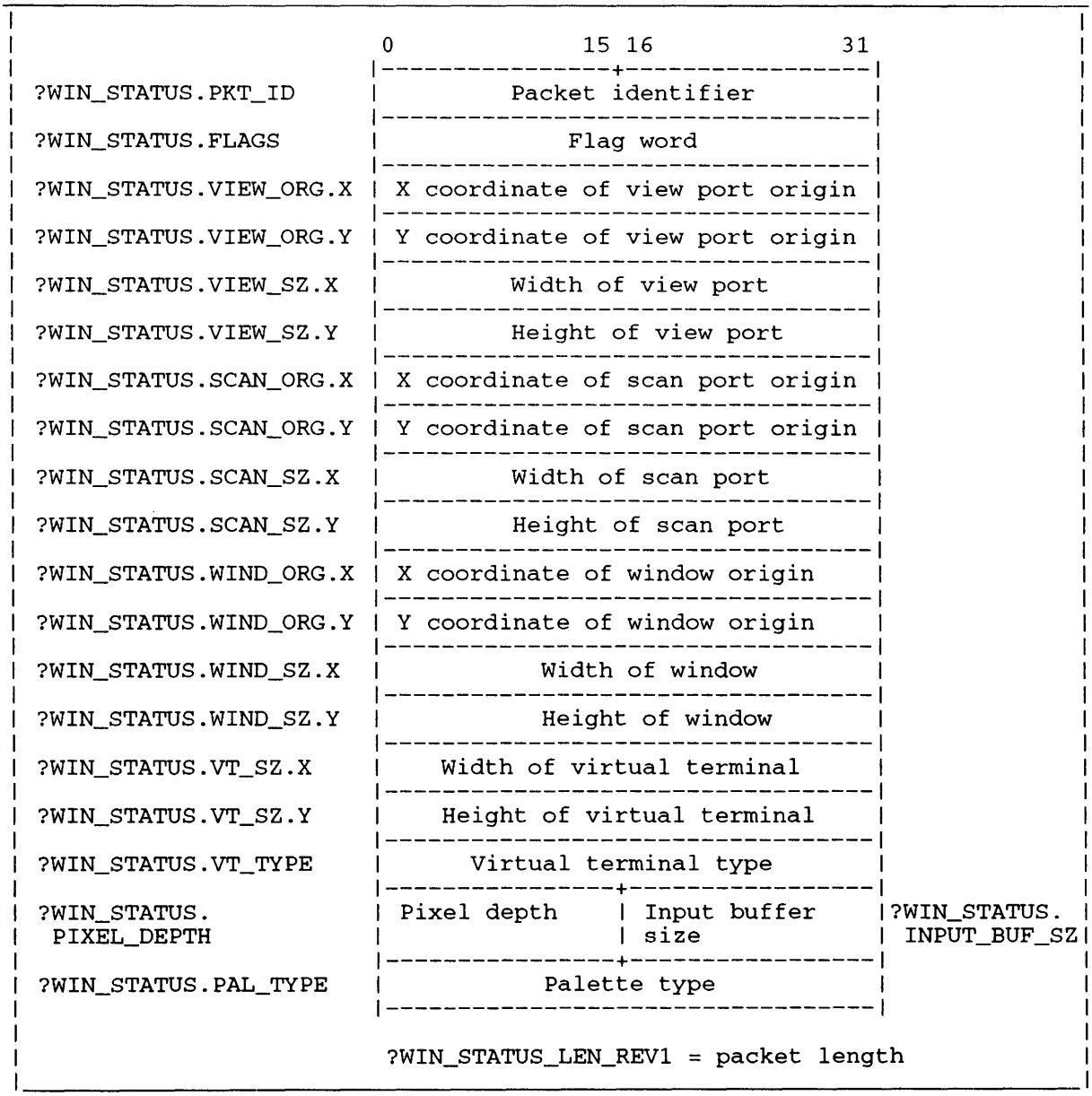


Figure 2–248. Structure of the ?WIN_WINDOW_STATUS Subpacket

Table 2–220. Contents of the ?WIN_WINDOW_STATUS Subpacket

Offset	Contents
?WIN_STATUS.PKT_ID (doubleword)	Packet identifier; set to ?WIN_STATUS_PKTID.
?WIN_STATUS.FLAGS (doubleword)	Flag word. Flag settings indicate the current status of window attributes, as follows: ?WIN_STATUS.FLAGS.GRP_HIDDEN - Window's group is invisible. ?WIN_STATUS.FLAGS.WIND_HIDDEN - Window is invisible. ?WIN_STATUS.FLAGS.GRP_SUSPENDED - Window's group is suspended. ?WIN_STATUS.FLAGS.WIND_SUSPENDED - Window is suspended. ?WIN_STATUS.FLAGS.KEYB_ENABLED - Keyboard input is enabled for this window. ?WIN_STATUS.FLAGS.PERM_ON - PERMANENCE is on for this window.
?WIN_STATUS.VIEW_ORG.X (doubleword)	X coordinate of the view port's origin (in pixels relative to the origin of the physical bit-mapped screen).
?WIN_STATUS.VIEW_ORG.Y (doubleword)	Y coordinate of the view port's origin (in pixels relative to the origin of the physical bit-mapped screen).
?WIN_STATUS.VIEW_SZ.X (doubleword)	Width of the view port (in pixels).
?WIN_STATUS.VIEW_SZ.Y (doubleword)	Height of the view port (in pixels).
?WIN_STATUS.SCAN_ORG.X (doubleword)	X coordinate of the scan port's origin (in characters for a character window, pixels for graphics window; relative to the origin of the virtual screen).
?WIN_STATUS.SCAN_ORG.Y (doubleword)	Y coordinate of the scan port's origin (in characters for a character window, pixels for a graphics window; relative to the origin of the virtual screen).
?WIN_STATUS.SCAN_SZ.X (doubleword)	Width of the scan port (in characters for a character window, pixels for a graphics window).
?WIN_STATUS.SCAN_SZ.Y (doubleword)	Height of the scan port (in characters for a character window, pixels for a graphics window).

(continued)

Table 2–220. Contents of the ?WIN_WINDOW_STATUS Subpacket

Offset	Contents
?WIN_STATUS.WIND_ORG.X (doubleword)	X coordinate of the window's origin, including window borders, if any. (In pixels, relative to the origin of the physical bit-mapped screen.)
?WIN_STATUS.WIND_ORG.Y (doubleword)	Y coordinate of the window's origin, including window borders, if any. (In pixels, relative to the origin of the physical bit-mapped screen.)
?WIN_STATUS.WIND_SZ.X (doubleword)	Width of the window, including borders, if any. (In pixels.)
?WIN_STATUS.WIND_SZ.Y (doubleword)	Height of the window, including borders, if any. (In pixels.)
?WIN_STATUS.VT_SZ.X (doubleword)	Width of the virtual terminal (in characters for a character window, pixels for a graphics window).
?WIN_STATUS.VT_SZ.Y (doubleword)	Height of the virtual terminal (in characters for a character window, pixels for a graphics window).
?WIN_STATUS.VT_TYPE (doubleword)	Type of virtual terminal for this window. ?WIN_VT_TYPE_D460 (Character window with 2 bits per pixel.) The virtual terminal is a software emulation of a D460 terminal, and supports the blink attribute. ?WIN_VT_TYPE_PIXMAP (Graphics window.) The virtual terminal is a pixel map.
?WIN_STATUS.PIXEL_DEPTH	Number of bits per pixel (for graphics windows).
?WIN_STATUS.INPUT_BUF_SZ	Size of the window's input buffer, in bytes.

(continued)

Table 2–220. Contents of the ?WIN_WINDOW_STATUS Subpacket

Offset	Contents
?WIN_STATUS.PAL_TYPE (doubleword)	Type of palette associated with the window. Palette types are ?WIN_PALETTE_TYPE_DEFAULT -- The system determines the palette type for this window. ?WIN_PALETTE_TYPE_USER -- User-definable palette with 2**(pixel depth) entries. For character windows, the pixel depth depends on the virtual terminal type. For graphics For graphics windows, the pixel depth is reported in offset ?WIN_STATUS.PIXEL_DEPTH. All entries in the palette are initially set to "dark". ?WIN_PALETTE_TYPE_SYSTEM -- System palette. The window shares the system palette, which is two entries long. The window cannot write to this palette. ?WIN_PALETTE_TYPE_SHARED -- Shared palette. The window shares the palette of the window that was specified in the main ?WINDOW packet when this window was created.

(concluded)

Getting Window IDs

You can issue ?WINDOW with function code ?WIN_GET_WINDOW_ID to find out the window ID number of a particular window. In the main packet, specify the pathname or channel number of the window you want. There is no subpacket for the ?WIN_GET_WINDOW_ID function; set ?WIN_PKT.SUBPKT to 0. The operating system returns the window ID number in offset ?WIN_PKT.WIND_ID.

Other functions of ?WINDOW let you get the window ID numbers for all windows in a particular window group, or for all windows on a terminal.

Getting the Status of a Physical Device

The ?WINDOW function ?WIN_DEVICE_STATUS returns the following information about a physical device:

- The dimensions of the physical pixel map.
- The pixel depth of the physical pixel map.
- The language ID of the physical keyboard.
- The type of pointer device in use (relative or absolute).

In the main packet, specify the window ID, pathname, or channel number of any window on the device you want. The operating system returns information in the subpacket. Figure 2–249 shows the structure of the subpacket; Table 2–221 details its contents.

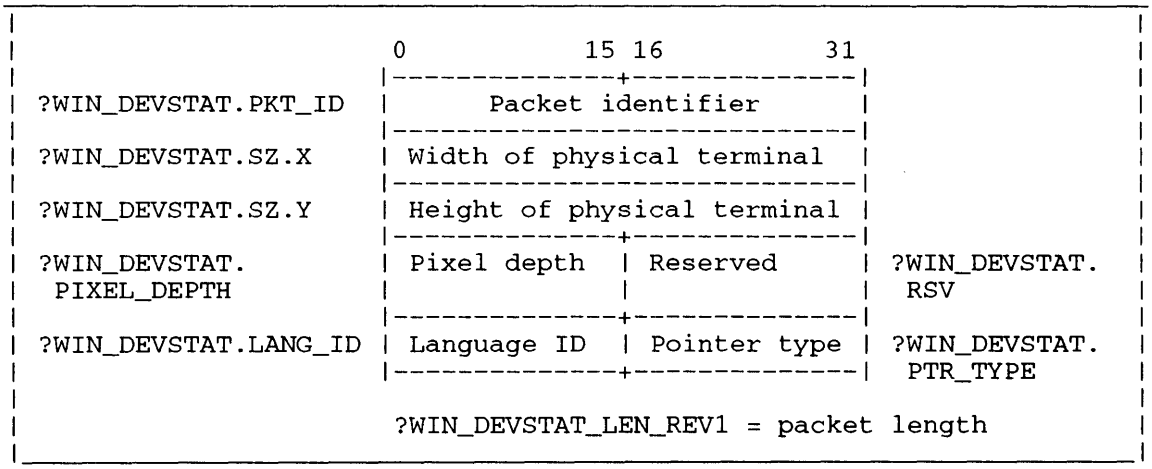


Figure 2–249. Structure of the ?WIN_DEVICE_STATUS Subpacket

Table 2–221. Contents of the ?WIN_DEVICE_STATUS Subpacket

Offset	Contents
?WIN_DEVSTAT.PKT_ID (doubleword)	Packet identifier; set to ?WIN_DEVSTAT_PKTID_REV1.
?WIN_DEVSTAT.SZ.X (doubleword)	Width of the physical screen (in pixels).
?WIN_DEVSTAT.SZ.Y (doubleword)	Height of the physical screen (in pixels).
?WIN_DEVSTAT. PIXEL_DEPTH	Number of bits per pixel in the physical graphics board.
?WIN_DEVSTAT.RSV	Reserved. (Set to 0.)
?WIN_DEVSTAT. LANG_ID	Language ID of the physical keyboard. Languages are ?WIN_LANG_ID_SWISS_GERMAN Swiss/German ?WIN_LANG_ID_CANADA_ENGLISH Canadian English ?WIN_LANG_ID_KATAKANA Katakana ?WIN_LANG_ID_ITALIAN Italian ?WIN_LANG_ID_CANADA_FRENCH Canadian French ?WIN_LANG_ID_US United States ?WIN_LANG_ID_UK United Kingdom ?WIN_LANG_ID_FRENCH French ?WIN_LANG_ID_GERMAN German ?WIN_LANG_ID_SWEDISH_FINNISH Swedish/Finnish ?WIN_LANID_SPANISH Spanish ?WIN_LANG_ID_DANISH_NORWEGIAN Danish/Norwegian
?WIN_DEVSTAT. PTR_TYPE	Type of pointer device attached to the physical terminal. Types are ?WIN_PTR_TYPE_RELATIVE -- A relative device, such as a mouse. The pointer moves as the device moves, but the actual position of the device does not affect the location of the pointer on the screen. ?WIN_PTR_TYPE_ABSOLUTE -- An absolute device, such as a bit pad or stylus. The position of the device on a grid directly determines the location of the pointer on the screen.

To Get Window IDs for Windows in a Group

To get a list of all window IDs in a particular window group, allocate an array of memory with at least as many entries as there are windows in the group. Then, issue the ?WINDOW call with function code ?WIN_RETURN_GROUP_WINDOWS; in the main packet, specify the pathname or channel number of any window in the group you want. In the subpacket, specify the number of entries you allocated in the array. If you specify fewer entries than there are windows in the group, the system returns the number of windows in offset ?WIN_RTN_GRPWINDS.O_NUM_WINDS, and takes the error return. Figure 2–250 shows the subpacket structure.

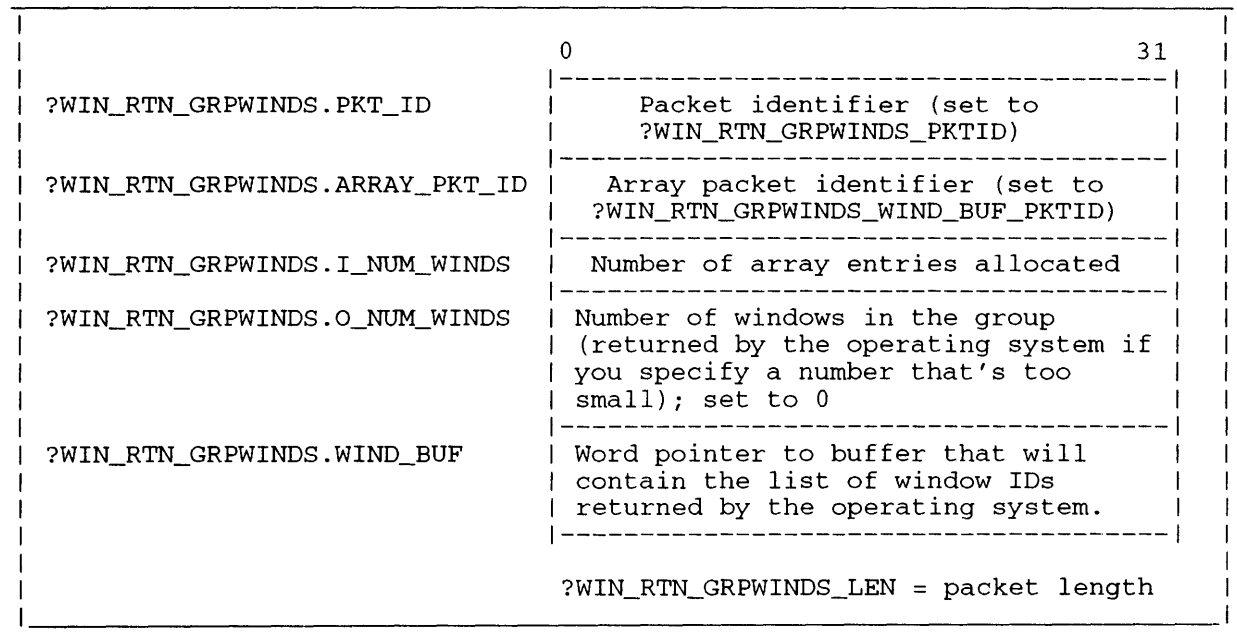


Figure 2–250. Structure of the ?WIN_RETURN_GROUP_WINDOWS Subpacket

To Get Window IDs for Windows on a Physical Terminal

To get a list of the window IDs for all the windows that exist on a particular physical terminal, allocate an array in memory, and issue the ?WINDOW call with function code ?WIN_RETURN_DEVICE_WINDOWS. In the main packet, specify the pathname or channel number of any window on the terminal you want. In the subpacket, specify how many doubleword entries you allocated in the array. If there are fewer entries in the array than there are windows on the terminal, the system returns the number of windows in offset ?WIN_RTN_DEVWINDS.O_NUM_WINDS, and takes the error return. Figure 2–251 shows the subpacket structure.

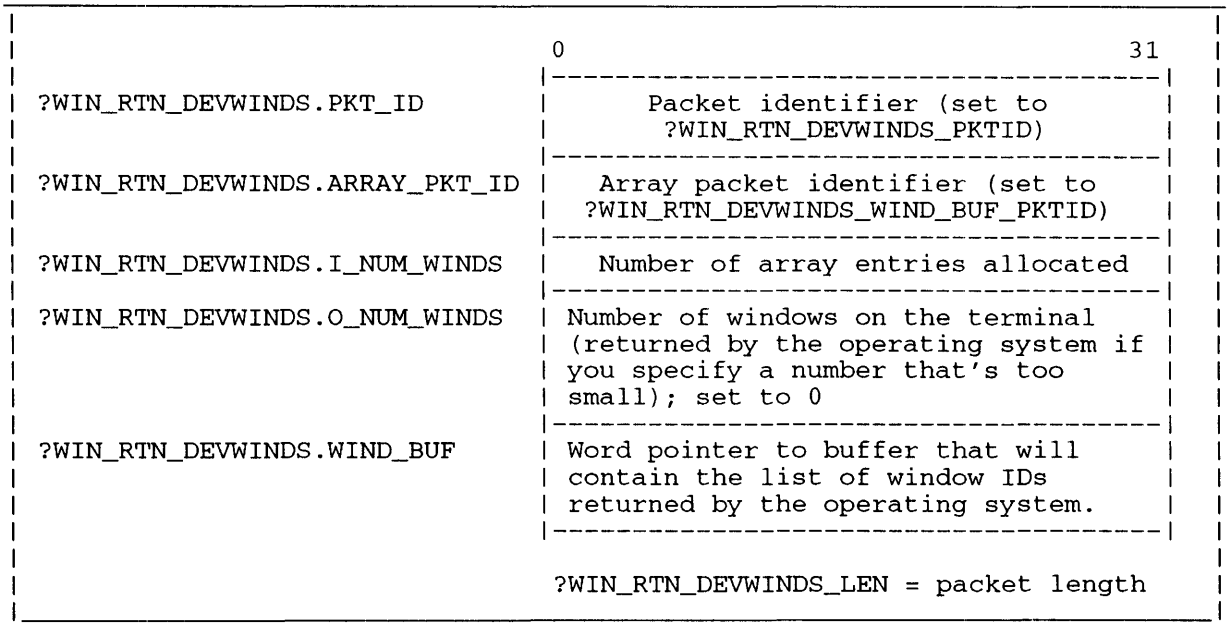


Figure 2-251. Structure of the ?WIN_RETURN_DEVICE_WINDOWS Subpacket

Notes

- See the description of ?GECHR in this chapter.

?WIRE
error return
normal return

Input

AC0	Starting address of the region to wire
AC1	Ending address of the region to wire
AC2	Reserved (Set to 0.)

Output

AC0	Unchanged
AC1	Unchanged
AC2	Undefined

Error Codes in AC0

ERAWM	Attempt to wire more pages than working set maximum
ERICM	Illegal system command (invalid call for 16–bit processes)
ERPRE	Invalid system call parameter
ERPRV	Caller not privileged for this action
ERPTY	Illegal process type
ERTPW	Too many pages wired. (The number of wired pages exceeds the limits of the swap file allocated to this process. Allowing another wired page would prevent swapping.)
ERVWP	Invalid word pointer passed as a system call argument

Why Use It?

?WIRE prevents the operating system from removing a range of logical pages from your working set. The pages that you wire with ?WIRE remain in your working set until you issue an ?UNWIRE call for them.

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

?WIRE permanently bindsthat is, wires) the region that you specify in AC0 and AC1 to the caller's working set. The operating system adjusts the caller's current working set size accordingly.

You can issue more than one ?WIRE against the same region, but subsequent system calls have no additional effect.

Before you issue ?WIRE, load AC0 and AC1 with the starting and ending addresses, respectively, of the region you want to wire. ?WIRE affects all pages within the region that you specify, including the pages that contain the starting and ending addresses. Note that this differs from the interpretation of the target region for ?UNWIRE.

There need not be a one–to–one correspondence between the ?WIRE and ?UNWIRE system calls that you issue. In other words, you can issue several ?WIRE system calls for different address regions, and then issue one ?UNWIRE system call to revoke all the previous ?WIRE system calls.

Notes

- See the description of ?UNWIRE in this chapter.

?WRB — Writes block I/O.

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

See the description of ?RDB in this chapter for more information on ?WRB.

?WRITE — Writes record I/O.

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

See the description of ?READ in this chapter for more information on ?WRITE.

?WRUDA — Writes a user data area (UDA).

?WRUDA
error return
normal return

See the description of ?RDUDA in this chapter for more information on ?WRUDA.

?WTSIG — Waits for a signal from another task or process.

?WTSIG
error return
normal return

Input

None

Output

AC0	Undefined
AC1	Unchanged
AC2	Undefined

Error Codes in AC0

ERSCI	System call at interrupt level

Why Use It?

?WTSIG, along with ?SIGWT and ?SIGNL, allows tasks to synchronize access to global memory that identical local servers share.

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

?WTSIG is similar to the IPC system call ?IREC. However, unlike ?IREC, ?WTSIG does not cause data movement. Instead, ?WTSIG waits for a signal (?SIGNL or ?SIGWT) from one of the following:

- Another task in the same process.
- Another task in a different process.

Warnings

The fast inter–process signaling mechanism is used by Data General supplied software, including (but not limited to) the PMGR interface routines found in the Agent in Ring 3 of all processes in the operating system. And, the system makes absolutely no checks as to the legitimacy of a ?SIGNL issued by anyone or to anyone.

These two warnings have the following implications.

If a task issues a ?SIGNL to another task, it must be absolutely sure that the next ?SIGWT or ?WTSIG issued by the receiving task will be the one it expects to receive the ?SIGNL. If such is not the case, the ?SIGNL may go to a ?SIGWT or ?WTSIG issued by another routine used by the task elsewhere in the process (e.g., the PMGR interface in Ring 3, or the INFOS® local server in Ring 4). There is no addressing of the message as is provided by the IPC message port number or the ?REC/?XMT mailbox location.

Then, after any task resumes execution after receipt of a ?SIGNL by ?WTSIG or ?SIGWT, it is incumbent upon the receiver to verify that the condition being waited for has in fact occurred. The operating system does not authenticate ?SIGNLs sent.

Notes

- See the descriptions of ?IREC, ?SIGNL, and ?SIGWT in this chapter.

?WTVERR Waits for a Virtual Timer Facility error message.

AOS/RT32 only

?WTVERR
error return
normal return

Input

AC0	Reserved (Set to 0.)
AC1	Reserved (Set to 0.)
AC2	Reserved (Set to 0.)

Output

AC0	Unchanged
AC1	VTF ID
AC1	Message code

Error Codes in AC0

This system call returns no error codes.

Why Use It?

Issue ?WTVERR to have the calling task monitor errors from the Virtual Timer Facility (VTF) driver routine. This is the only way you can detect overrun errors.

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 pends the calling task until the VTF driver posts an error message to the VTF error queue. Each issuance of ?WTVERR removes an error message from the queue and the VTF driver posts error messages in a first–in first–out manner; consequently, ?WTVERR always returns the oldest error message. Exception: error ?VTFHUNG is always at the front of the queue.

The error messages consist of two 32–bit numbers. The first number is returned in AC1 and is the VTF ID. The second number is returned in AC2 and is one of the following.

?VTERIOVR Interrupt–level code overrun

?VTERSOVR Signal mode overrun

?VTFHUNG VTF driver routine is hung

?VTERIOVR indicates that an interrupt–level code path did not return by the time the VTF driver routine scheduled it to run next.

?VTERSOVR indicates that the VTF driver sent a VTF signal to the task when a VTF signal was already outstanding.

?VTERIOVR and ?VTERSOVR together indicate that the VTF driver has suspended the VTF timer, although this driver allowed the interrupt–level code path (for ?VTERIOVR) to run to completion one last time.

?VTFHUNG indicates that the VTF driver is "hung." The cause is probably an interrupt–level code path's hanging — although other OS problems can generate this error as a side effect. Consider the VTF driver "hung" when it receives a preset number of timer interrupts and it cannot return. This preset number is specified during the generation of your OS.

When the VTF driver hangs, it aborts all nested VTF interrupt–level code paths in progress, suspends the VTF timer for each of these code paths, forces error message ?VTFHUNG for each VTF timer thus suspended, and dismisses interrupts. Note that this situation results in the termination of interrupt–level code paths asynchronously, at any point in their execution, without their completing.

NOTE: The OS sends errors ?VTERIOTR and ?VTERSOTR according to the setup mode of the VTF timer that you specified in the packet for system call ?VTFCREATE. Accordingly a VTF timer can ignore overrun errors, in which case you will never see an error in the VTF error queue, or else the timer can send an error only after the OS detects some fixed number of consecutive errors. Information about the error mode of the VTF timer that caused the error is *not* sent to ?WTVERR. So, if you need this information you must retain it from the original ?VTFCREATE system call for that timer.

?WTVSIG — Waits for a Virtual Timer Facility signal.

AOS/RT32 only

?WTVSIG
error return
normal return

Input

AC0	Reserved (Set to 0.)
AC1	Reserved (Set to 0.)
AC2	Reserved (Set to 0.)

Output

AC0	Unchanged
AC1	Number of signals sent
AC2	Unchanged

Error Codes in AC0

None

Why Use It?

You issue ?WTVSIG to wait for a signal from the VTF driver routine. This signal indicates that a periodic event has occurred.

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

?WTVSIG pends the calling task if a signal is not already pending from the Virtual Timer Facility (VTF). If a signal is already pending, ?WTVSIG returns immediately with the number of pending signals in AC1. This number equals the number of signals since the last return from ?WTVSIG by this task. The number informs the calling task of any missed signals so that this task can take any required actions.

Each VTF timer that is set up to send signals is associated with a specific task. Furthermore, only one VTF timer may associate with a specific task at any given time. Thus a task receives only signals directed at it. However, any task may issue ?WTVSIG at any time, regardless of whether a VTF timer exists to send signals to the task.

System call ?VTFCREATE, which creates a VTF timer, requires the existence of the task assigned as the target of the VTF timer's signals. Consequently you must create the waiting task before the VTF timer can send signals.

NOTE: ?WTVSIG responds only to VTF signals. In particular, it does not respond to system call ?SIGNL even if you issue ?SIGNL against the same task that is pended on ?WTVSIG. The ?SIGNL/?WTSIG signal system is completely separate from the VTF signal system.

AOS/VS II only

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

Input

AC0	Reserved (set to 0)
AC1	Reserved (set to 0)
AC2	The system call packet address, unless you specify the address as an argument to ?XCREATE.

Output

AC0	Unchanged or an error code
AC1	Unchanged
AC2	Unchanged

Error Codes in AC0

ERPKT Illegal packet id
ERRVN Reserved value not zero
ERPRV Caller not privileged for this action
ERIVP Invalid port number specified
ERLVL Maximum directory tree depth exceeded
ERNAE Filename already exists
ERWAD Write access denied
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_CPD_MINIMUM_SPACE_REQUIREMENT
You specified an invalid control point directory size.
ER_FS_FILE_CLASS_NOT_IMPLEMENTED
You specified an invalid packet ID.
ER_FS_INVALID_ACL_BYTE_PTR
You specified an invalid byte pointer to the ACL list.
ER_FS_INVALID_CREATE_FILE_TYPE
You specified an invalid file type.
ER_FS_INVALID_ELEMENT_SIZE
You specified an invalid element size.
ER_FS_INVALID_MAX_NUM_INDEX_LEVELS
You specified an invalid maximum number of index levels.
ER_FS_INVALID_PATHNAME_BYTE_PTR
You specified an invalid byte pointer to the pathname.
ER_FS_INVALID_PRIMARY_EMEM_COUNT
You specified an invalid primary element count.
ER_FS_INVALID_PKT_PTR
You specified an invalid pointer to the system call packet.
ER_FS_INVALID_TIME_BLOCK_PTR
You specified an invalid word pointer to the time–block subpacket.
ER_FS_INVALID_XCREATE_FUNCTION
You specified an invalid function code.

?XCREATE Continued

Why Use it?

You should use ?XCREATE to create new files and directories for input/output. ?XCREATE lets you specify default element sizes for data and indexes. Files can also be created 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 directory of the target file.

What It Does

?XCREATE is an extension of ?CREATE and allows further flexibility in specifying the characteristics of a file. ?XCREATE creates a file or directory based on the file type and attributes you specify in the packet.

The ?XCREATE packet consists of a main packet and one or more subpackets. The main packet (Figure 2–252) specifies parameters common to all of the file types. Table 2–222 describes the contents of the main packet. The subpackets hold information specific to a file type. The remaining sections describe the structures of the subpackets.

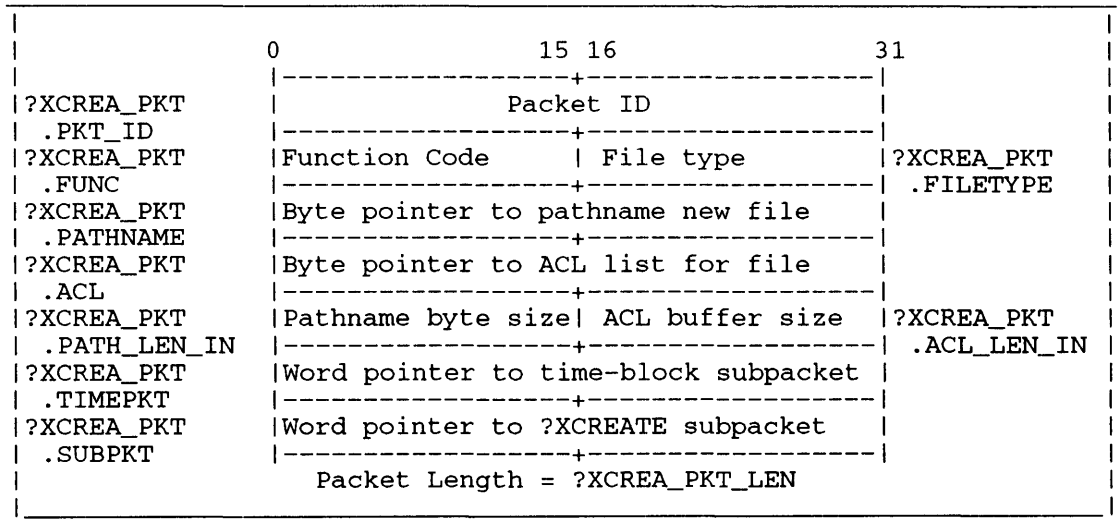


Figure 2–252. Structure of ?XCREATE Main Packet

Table 2–222. Contents of ?XCREATE Main Packet*

Offset	Contents
?XCREA_PKT .PKT_ID	Packet Identifier (Set to ?XCREA_PKT_PKTID.)
?XCREA_PKT .FUNC	The function code identifies the subpacket to use and is based upon the type of file to create. Use one of the following function codes: ?XCREA_OTH_SUBCODE for other subpackets ?XCREA_DIR_SUBCODE for directory subpacket ?XCREA_IPC_SUBCODE for IPC subpacket ?XCREA_LNK_SUBCODE for link subpacket

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

Table 2-222. Contents of ?XCREATE Main Packet*

Offset	Contents
?XCREA_PKT .FILETYPE	File type; see Table 2-223. Except for certain peripheral devices, you can create most of the files.
?XCREA_PKT .PATHNAME	Byte pointer to pathname of new file to create. The pathname is resolved down to the filename entry to determine the parent directory.
?XCREA_PKT .ACL	Byte pointer to the optional access control list. If you set it to 0, the new file has a null ACL list. (Set it to 0 for a link file.) The ACL buffer format is: username<0><accesstype+[...]><0> where, accesstype is one of the following: ?FACO Owner access ?FACW Write access ?FACA Append access ?FACR Read access ?FACE Execute access The following example assigns JEFF an OWARE access and LISA an RE access. JEFF<0><?FACO+?FACW+?FACA+?FACR+?FACE> LISA<0> <?FACR+?FACE><0> DEFAULT = -1, assign the caller's default ACL).
?XCREA_PKT .PATH_LEN_IN	Number of bytes in pathname, including null delimiter.
?XCREA_PKT .ACL_LEN_IN	If you set ?XCREA_PKT.ACL to point to a buffer containing the ACL string, you must also set ?XCREA_PKT.ACL_LEN_IN to the number of bytes in the ACL string. Otherwise, set it to zero.
?XCREA_PKT .TIMEPKT	Word pointer to the optional time-block subpacket. If you want to set the creation, last access, and last modify times of the file, set ?XCREA_PKT.TIMEPKT to point to a time packet. See Figure 2-253 for the structure of the time subpacket. DEFAULT = -1 (Set all values to current date and time).
?XCREA_PKT .SUBPKT	Word pointer to the ?XCREATE subpacket

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

Table 2–223. Valid ?XCREATE File Types

Type	Meaning	Comments
?FCPD	Control Point Directory	Use ?XCREA_DIR_SUBCODE.
?FDIR	Disk Directory	Use ?XCREA_DIR_SUBCODE.
?FFCC	FORTRAN Carriage Control	Use ?XCREA_OTH_SUBCODE.
?FGLT	Generic Labeled Tape	Use ?XCREA_IPC_SUBCODE.
?FIPC	IPC Port Entry	Standard IPC file. Use ?XCREA_IPC_SUBCODE.
?FLCC	FORTRAN Carriage Control	Use ?XCREA_OTH_SUBCODE.
?FLNK	Link File	Use ?XCREA_LNK_SUBCODE.
?FNCC	FORTRAN Carriage Control	Use ?XCREA_OTH_SUBCODE.
?FOCC	FORTRAN Carriage Control	Use ?XCREA_OTH_SUBCODE.
?FPIP	Pipe File	The element size must be a multiple of 4. Use ?XCREA_OTH_SUBCODE.
?FPRG	AOS Program File	Program file for use under AOS (16-bit code). Use ?XCREA_OTH_SUBCODE.
?FPRV	AOS/VS Program File	Program file for use under AOS/VS (32- or 16-bit code). Use ?XCREA_OTH_SUBCODE.
?FQUE	Queue Entry	Use ?XCREA_IPC_SUBCODE.
?FSDF	System Data File	You cannot specify record format types (?XCREA_OTH.REC_FORM) for system data files. Use ?XCREA_OTH_SUBCODE.
?FSPR	Spoolable Peripheral Directory	Special IPC file. Use ?XCREA_IPC_SUBCODE.
?FSTF	Symbol Table File	Produced by the Link utility and used primarily by the OS. Use ?XCREA_OTH_SUBCODE.
?FTXT	Text File	Should contain ASCII text. Use ?XCREA_OTH_SUBCODE.
?FUDF	User Data File	Usually applies to object files. Use ?XCREA_OTH_SUBCODE.
?FUNX	MV/UX File	File for use under MV/UX. Use ?XCREA_OTH_SUBCODE.
?FUPF	User Profile File	Used by PREDITOR (user profile editor) and EXEC. Use ?XCREA_OTH_SUBCODE.

Time Subpacket

The time subpacket gives you the option of setting the times for file creation, last access, or last modification. You must express the time–block values as single–precision integers. You state the date in days from December 31, 1967 to the date the file was created, accessed or modified. You specify the time as the number (less than 43,200 decimal) of biseconds (half the number of seconds) since midnight. Figure 2–253 shows the time subpacket. Table 2–224 describes the contents of the subpacket.

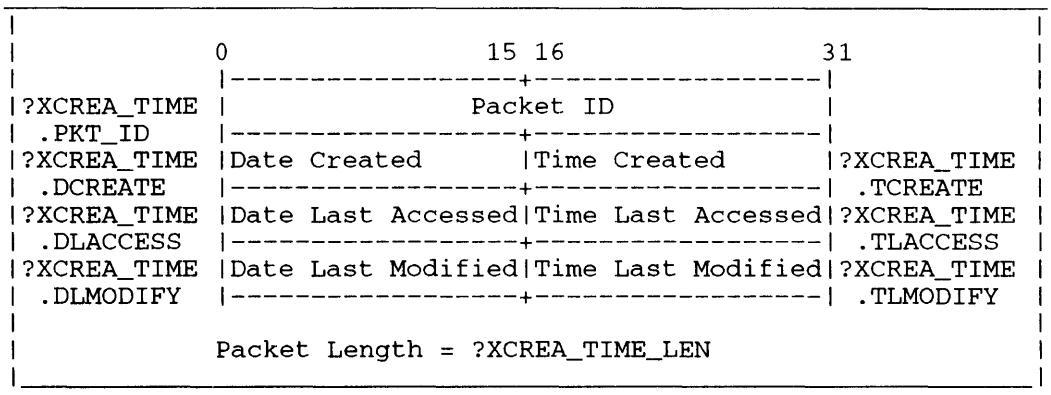


Figure 2–253. Structure of ?XCREATE Time–Block Subpacket

Table 2–224. Contents of ?XCREATE Time–Block Subpacket

Offset	Contents
?XCREA_TIME .PKT_ID	Time subpacket identifier. (Set to ?XCREA_TIME_PKTID.)
?XCREA_TIME .DCREATE	Date file was created.
?XCREA_TIME .TCREATE	Time file was created.
?XCREA_TIME .DLACCESS	Date file was last accessed.
?XCREA_TIME .TLACCESS	Time file was last accessed.
?XCREA_TIME .DLMODIFY	Date file was last modified.
?XCREA_TIME .TLMODIFY	Time file was last modified.

Other Subpacket

Figure 2–254 shows the subpacket to create all file types, except for a directory, a link, or an IPC file. Table 2–225 describes the contents of the subpacket.

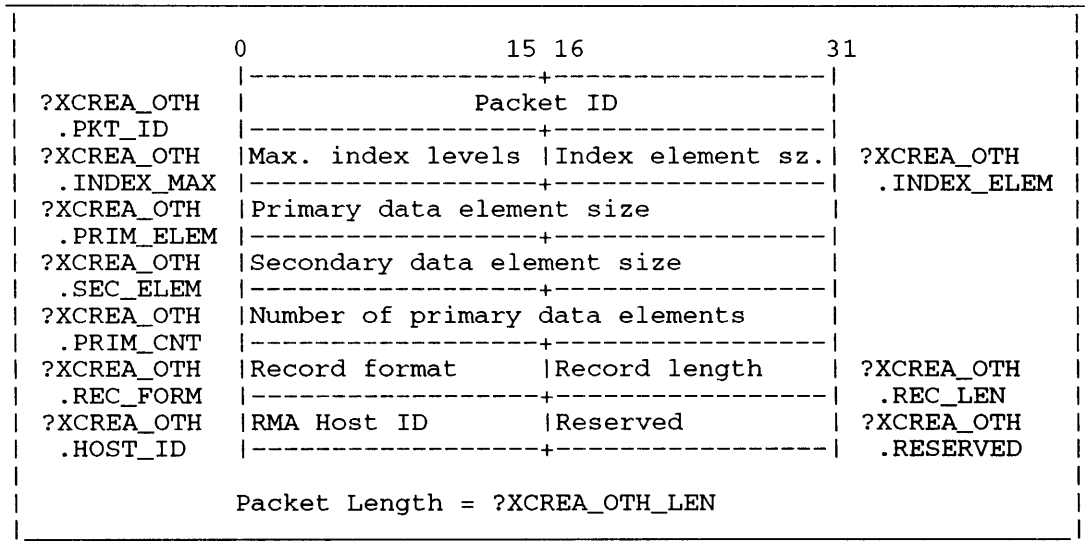


Figure 2–254. Structure of ?XCREATE Other Subpacket

Table 2–225. Contents of ?XCREATE Other Subpacket

Offset	Contents
?XCREA_OTH .PKT_ID	Other Subpacket Identifier (set to ?XCREA_OTH_PKTID.)
?XCREA_OTH .INDEX_MAX	Maximum number of index levels for new file. (Range 0–3.) Default = -1; assign same default as specified for LDU.
?XCREA_OTH .INDEX_ELEM	Index element size. (Range 1–255.) Default = -1; assign same default as specified for LDU.
?XCREA_OTH .PRIM_ELEM	Primary data element size. (Range 1–2147483647.) Default = -1; assign same default as specified for LDU.
?XCREA_OTH .SEC_ELEM	Secondary data element size. (Range from 1–2147483647.) Default = -1; assign same default as specified for LDU.
?XCREA_OTH .PRIM_CNT	Number of primary data elements to allocate before using secondary data element. Default = -1; assign same default as specified for LDU.

(continued)

Table 2–225. Contents of ?XCREATE Other Subpacket

Offset	Contents
?XCREA_OTH .REC_FORM	Record format. Can be one of the following masks: ?XCREA_RECFRM_DYNAMIC — Dynamic ?XCREA_RECFRM_DATA_SEN — Data-sensitive ?XCREA_RECFRM_FIX_LEN — Fixed-length ?XCREA_RECFRM_VAR_LEN — Variable-length ?XCREA_RECFRM_UNDEF_LEN — Undefined-length ?XCREA_RECFRM_BLK_REC_VAR — Variable block and record
?XCREA_OTH .REC_LEN	Record length, for fixed-length record formats only. (Otherwise, set to 0.)
?XCREA_OTH .HOST_ID	Host identifier. (Set to 0; applies to RMA file only.)
?XCREA_OTH .RESERVED	Reserved. (Set to 0.)

(concluded)

Directory Subpacket

Figure 2–255 shows the directory subpacket. Table 2–226 describes the contents of the subpacket.

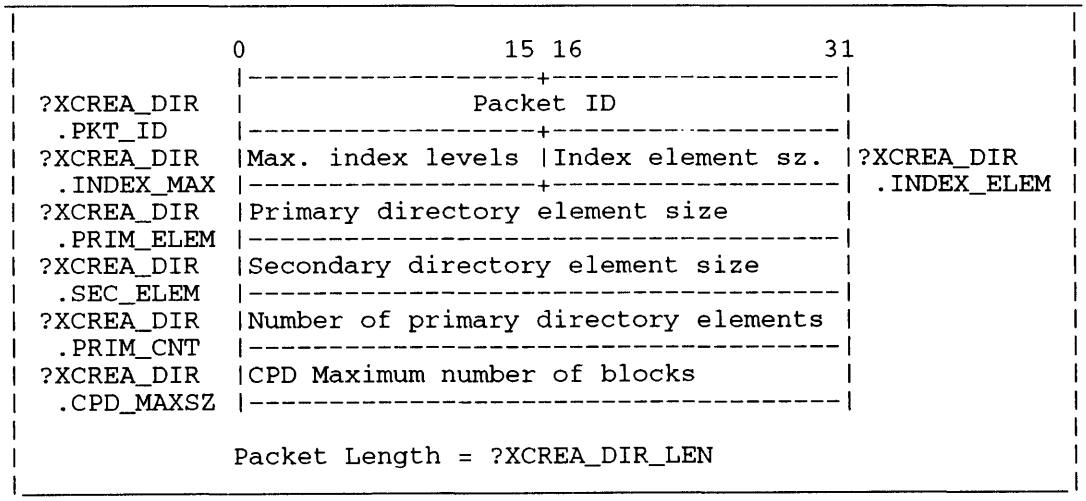


Figure 2–255. Structure of ?XCREATE Directory Subpacket

Table 2–226. Contents of ?XCREATE Directory Subpacket

Offset	Contents
?XCREA_DIR .PKT_ID	Directory Subpacket Identifier (Set to ?XCREA_DIR_PKTID.)
?XCREA_DIR .INDEX_MAX	Maximum index levels (Range 0-3). Default = -1; assign same default as specified for LDU. If 0, build a contiguous directory.
?XCREA_DIR .INDEX_ELEM	Index element size. (Range 1-255.) Default = -1; assign same default as specified for LDU.
?XCREA_DIR .PRIM_ELEM	Primary directory element size. (Range 1-2147483647). Default = -1; assign same default as specified for LDU.
?XCREA_DIR .SEC_ELEM	Secondary directory element size. (Range 1-2147483647). Default = -1; assign same default as specified for LDU.
?XCREA_DIR .PRIM_CNT	Number of primary directory elements to allocate before using secondary data element. Default = -1; assign same default as specified for LDU.
?XCREA_PKT .CPD_MAXSZ	Maximum number of disk blocks available to a control point directory and all of its subordinate files, excluding files in a subordinate LDU. Must be at least 1 for ?FCPD. Must be 0 for ?FDIR. To change the value, use the ?CPMAX call. Default = -1; largest maximum assigned.

IPC Subpacket

One process uses an interprocess communication file to send messages to another process. With the ?FIPC file type, the operating system creates a standard interprocess communication file in the caller's initial working directory. With the ?FSPR file type, the operating system creates a special IPC file for a spoolable peripheral device.

When the process that created the IPC file terminates, the operating system deletes the IPC file. Figure 2–256 shows the IPC subpacket. Table 2–227 describes the contents of the subpacket.

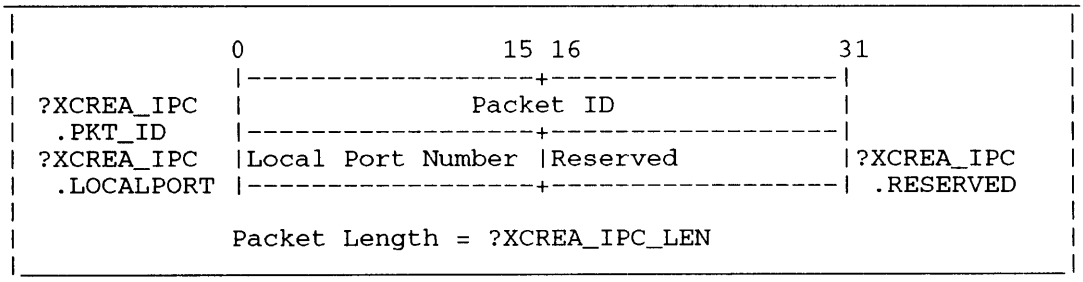


Figure 2–256. Structure of ?XCREATE IPC Subpacket

Table 2–227. Contents of ?XCREATE IPC Subpacket

Offset	Contents
?XCREA_IPC .PKT_ID	IPC Subpacket identifier (Set to ?XCREA_IPC_PKTID.)
?XCREA_IPC .LOCALPORT	IPC Local Port Number. (Range 1-?MXLPN.) ?XCREA_IPC.LOCALPORT has no default value.
?XCREA_IPC .RESERVED	Reserved (Set to 0.)

Link Subpacket

Use the link subpacket to create a link file. Specify the pathname to the link file in the main packet. Place the byte pointer to a buffer containing the link resolution pathname in this subpacket. The resolution pathname is not resolved when the link file is created. Therefore, no error is returned if the resolution pathname does not exist or contains illegal characters. Figure 2–257 shows the link subpacket. Table 2–228 describes the contents of the subpacket.

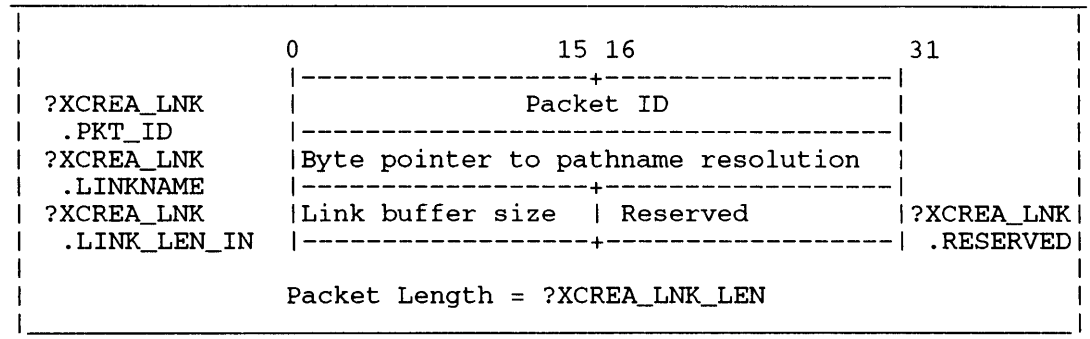


Figure 2–257. Structure of ?XCREATE Link Subpacket

Table 2–228. Contents of ?XCREATE Link Subpacket

Offset	Contents
?XCREA_LNK .PKT_ID	Link Subpacket identifier (Set to ?XCREA_LNK_PKTID.)
?XCREA_LNK .LINKNAME	Byte pointer to link resolution pathname.
?XCREA_LNK .LINK_LEN_IN	Byte size of buffer containing link pathname.
?XCREA_LNK .RESERVED	Reserved (Set to 0).

16–Bit System Call Support

The ?XCREATE system call supports a 16–bit version of the call, which uses the same packet offset as 32–bit call. The 16–bit version also requires a ring field with the 16–bit addresses.

Example of Text File Creation

In this example (Figures 2–258 and 2–259), the caller wants to create a text file, BRIANNE_MEMO, in the working directory. The primary and secondary element sizes are 16 and 4, respectively. All other assignments are defaulted. The main packet and the other subpacket should be filled in as follows:

Set AC0 = 0
Set AC1 = 0
Set AC2 = word pointer to main packet.

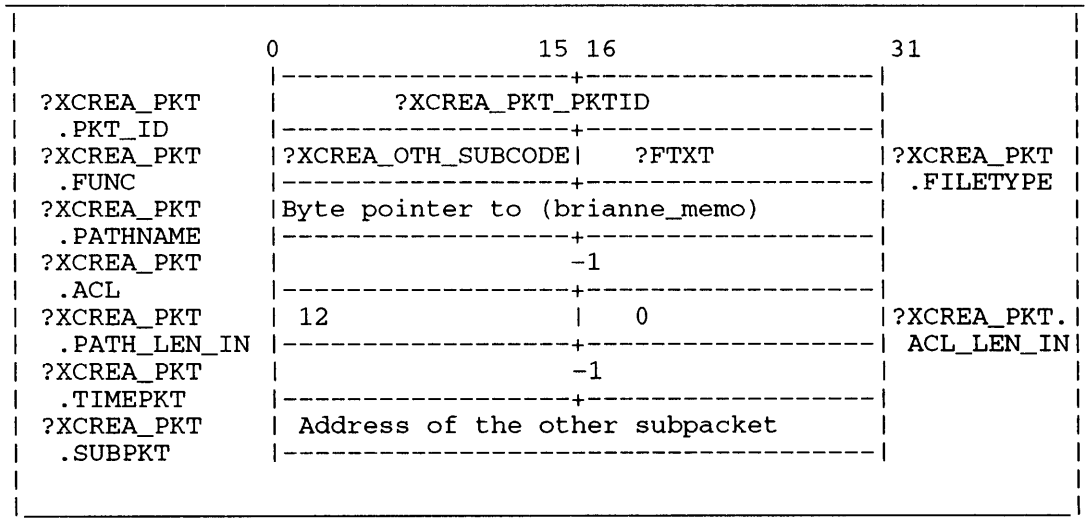


Figure 2–258. Example of ?XCREATE Main Packet for TXT File

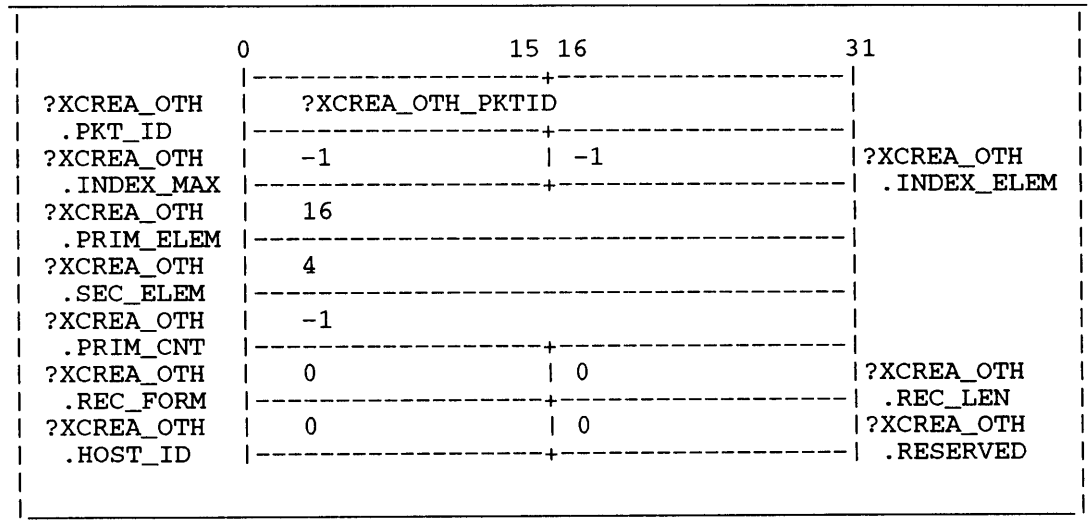


Figure 2–259. Example of ?XCREATE Other Subpacket for TXT File

The call assigns the default values for the maximum index level, index element size, and number of primary elements that were specified for the Logical Disk Unit (LDU).

Example of Directory File Creation

In this example (Figures 2–260 through 2–262), the caller wants to create a directory named TSAO in :UDD:BRIANNE. The caller would like the directory to grow to a maximum of two index levels with each index element holding three blocks. The caller also wants the primary directory element size to be 2 blocks and allow up to 2 primary elements to be allocated before going to the secondary directory element of size 1 block. In addition, the caller wants to specifically set the times for 2:14:50 on August 12, 1988. The ?XCREATE call should be setup as follows:

Set AC0 = 0
Set AC0 = 0
Set AC2 = Word pointer to main packet

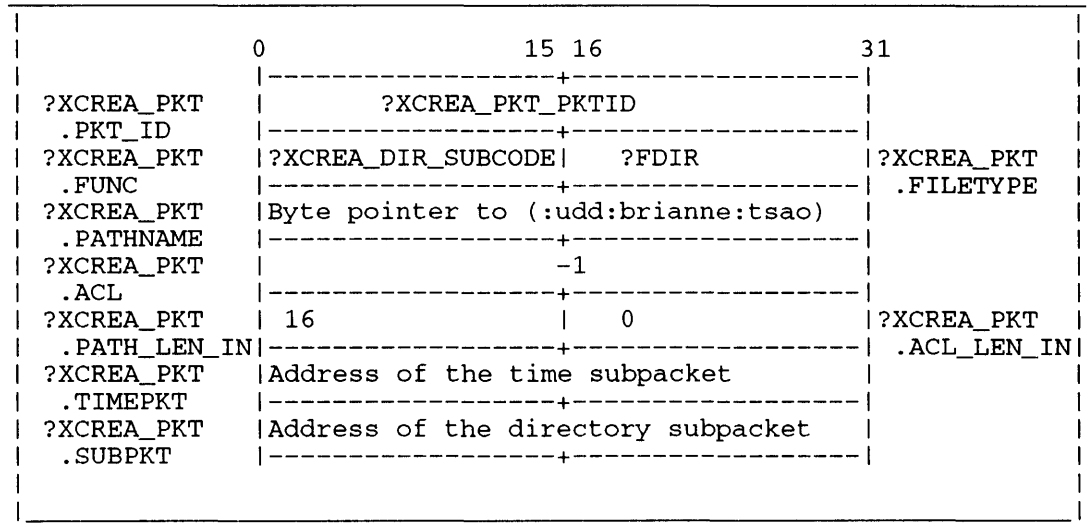


Figure 2–260. Example of ?XCREATE Main Packet for DIR File

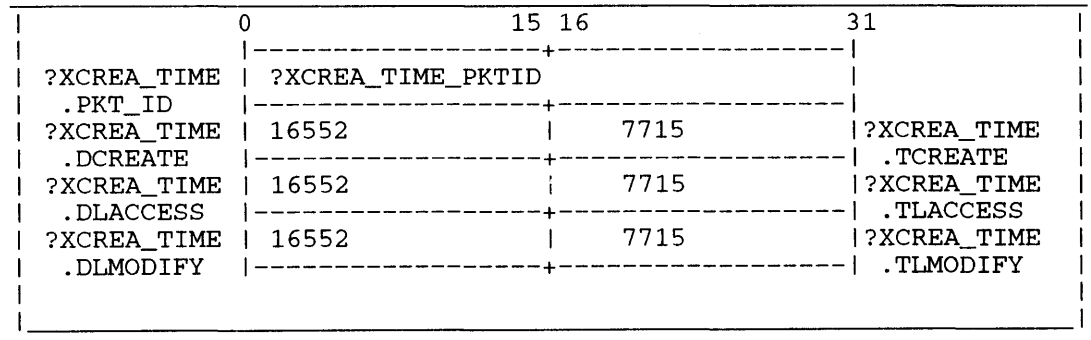


Figure 2–261. Example of ?XCREATE Time Subpacket for DIR File

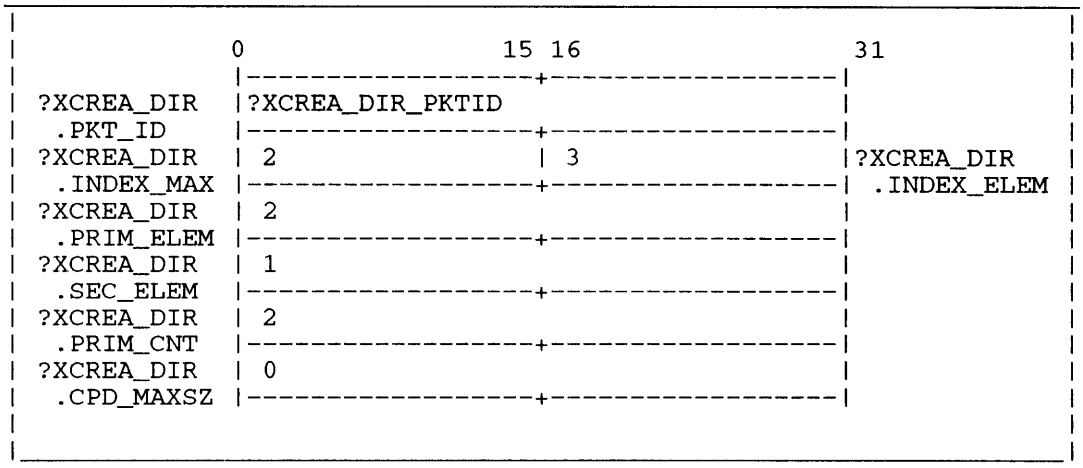


Figure 2–262. Example of ?XCREATE Directory Subpacket for DIR File

Example of IPC File Creation

In this example (Figures 2–263 and 2–264), the caller wants to create an IPC file named, :UDD:BRIANNE:MY_IPC_PORT, to send and receive IPC messages. The caller wants all messages to go through local port, 7. The caller assigns the ACL to be BRIANNE OWARE with a byte size of 12. The caller sets the packets as follows:

Set AC0 = 0
Set AC1 = 0
Set AC2 = Word pointer to main packet

The ACL string format looks like: BRIANNE<0><?FACO+?FACW+?FACA+?FACR+?FACE><0>

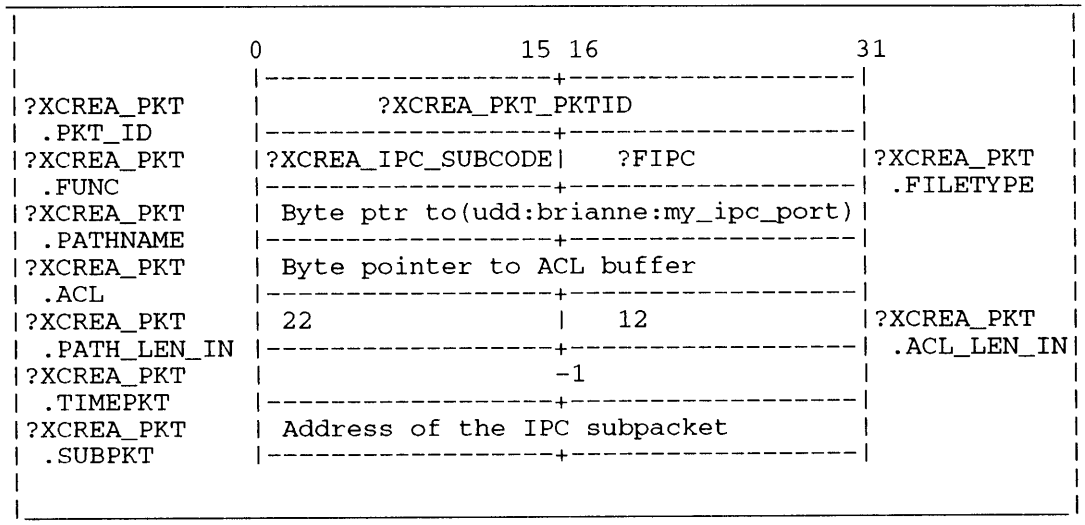


Figure 2–263. Example of ?XCREATE Main Packet for IPC File

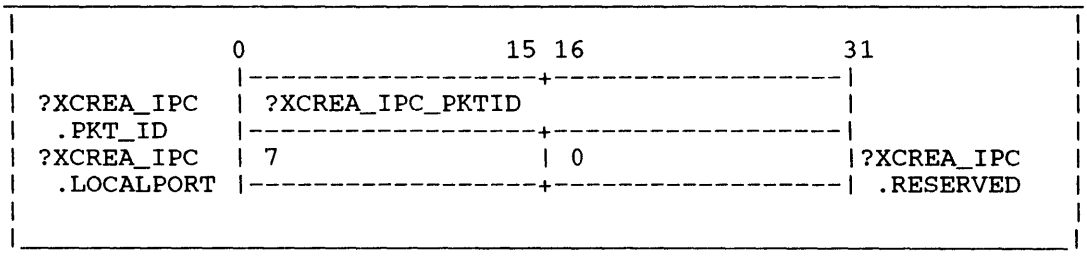


Figure 2–264. Example of ?XCREATE IPC Subpacket for IPC File

Example of Link File Creation

In this example (Figures 2–265 and 2–266), the caller wants to create a link file to, :UDD:BRIANNE:BRIANNE_MEMO, from his working directory, :UDD:DOUGLAS. The caller names the link filename, NOT_MY_MEMO. The caller sets the packets as follows:

Set AC0 = 0
Set AC1 = 0
Set AC2 = Word pointer to main packet

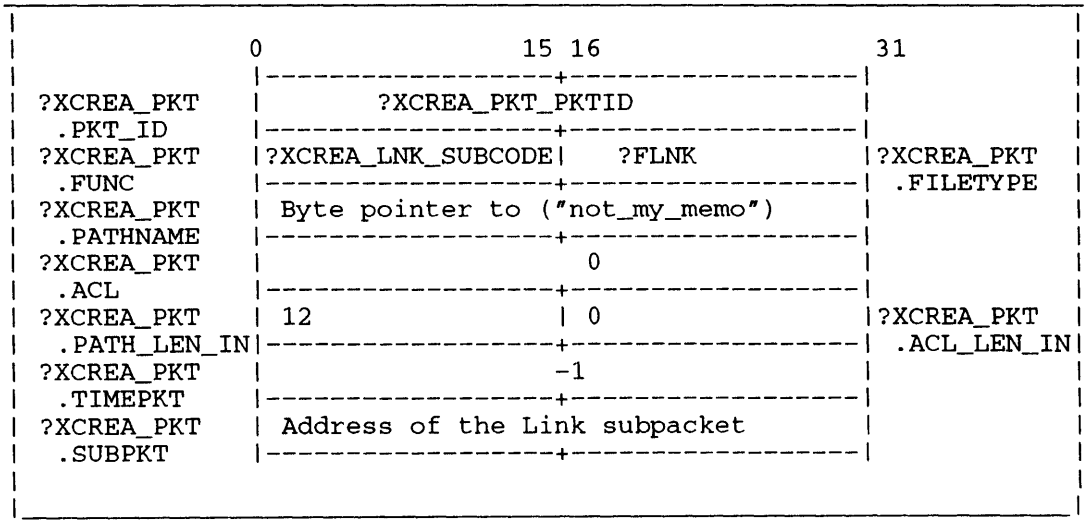


Figure 2–265. Example of ?XCREATE Main Packet for LNK File

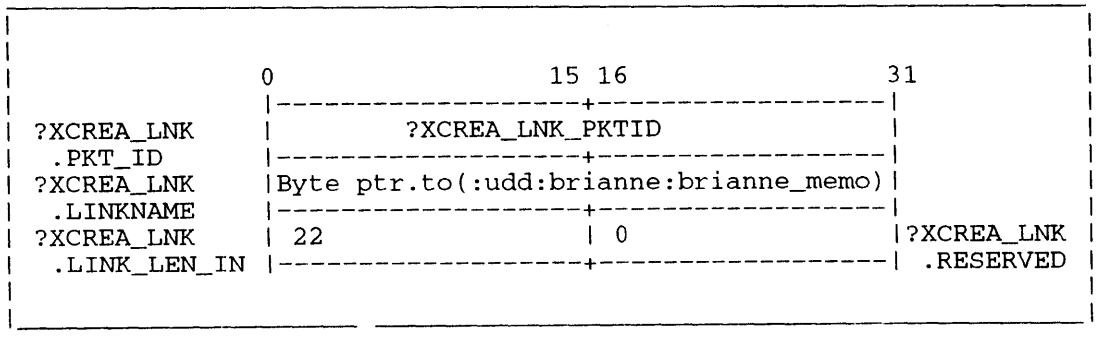


Figure 2–266. Example of ?XCREATE Link Subpacket for LNK File

AOS/VS II only

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

Input

AC0	Reserved (set to 0)
AC1	Reserved (set to 0)
AC2	The system call packet address, unless you specify the address as an argument to ?XFSTAT.

Output

AC0	Unchanged or an error code
AC1	Unchanged
AC2	Unchanged

Error Codes in AC0

ERPKT Illegal packet id
ERRVN Reserved value not zero
ERFAD File access denied
ERCIU Channel in use
ERFNO Channel not open
ERICN Illegal channel
ERVWP Invalid word pointer passed as a system call argument
ERVBP Invalid byte pointer passed as system call argument
ERUOL Physical Unit offline
ER_FS_INVALID_XFSTAT_FUNCTION
The ?XFSTAT call does not support subpackets and you specified one. The ?XFSTAT_PKT.FUNC must be set to zero (AOS/VS II only)
ER_FS_DIRECTORY_NOT_AVAILABLE
Directory not available because the LDU was force released (AOS/VS II only)
ER_FS_DIR_DOES_NOT_EXIST
The operating system cannot find the directory
ER_FS_FILE_CLASS_NOT_IMPLEMENTED
You specified an invalid packet ID
ER_FS_FILENAME_DOES_NOT_EXIST
The operating system cannot find the file
ER_FS_ILLEGAL_FILENAME_CHAR
You specified a filename with an illegal character
ER_FS_INVALID_PACKET_REV
The ?XFSTAT packet does not match the current revision of the operating system
ER_FS_INVALID_PATHNAME_BYTE_PTR
You specified an invalid byte pointer to the pathname
ER_FS_INVALID_XFSTAT_OPTIONS
You specified an invalid ?XFSTAT_PKT.OPTIONS flag
ER_FS_SAM_DISK_CURRENTLY_OPEN_EXCLUSIVE
The unit is exclusively opened by another process
ER_FS_ZERO_LENGTH_FILENAME
The specified file contains zero bytes

Why Use It?

Similar to ?FSTAT, this call determines the attributes assigned to a file or directory that you created with ?CREATE or ?XCREATE. The call also returns the current size and structure of the file, the date and time it was last accessed or modified, and the counter statistics. You should use ?XFSTAT to obtain attributes assigned to files created under the new file system.

Who Can Use It?

There are no special process privileges needed to issue the call. The caller must meet one of the following access rules to get a file's status:

- Read and Execute access to the file's parent directory when no access to the file is granted.
- Execute access to the file's parent directory and any access to the file.

What It Does

The ?XFSTATall is an extension of ?FSTAT. The ?XFSTAT call returns the file statistics to the file status packet you define. The ?XFSTAT call uses one packet to hold input and output information for all file types.

To use the ?XFSTAT call, you must supply the packet identification, the channel number or byte pointer, and set the channel number or pathname option flag. Set all unused input flags and fields to zero. (The file system verifies them to be zero.) To interpret the output after the call completes, begin with the file type.

The ?XFSTAT call uses a common set of input packet offsets for all of the file types. The input packet offsets are unchanged upon completion of the call. The format of the output packet offsets vary according to the file type. The ?XFSTAT call returns one of four packet formats: IPC, directory, other, and unit. The file system sets any undefined output field to zero. We show these fields as blank fields in the figures. Table 2–229 lists each file type and the associated output packet format that ?XFSTAT returns.

Table 2–229. Valid ?XFSTAT File Types

File Type	Meaning	Output Packet Format
?FAFI	APL file	Other packet
?FAWS	APL workspace file	Other packet
?FBBS	Business BASIC save file	Other packet
?FBCI	Basic core image	Other packet
?FCON	Console	IPC packet
?FCPD	Control Point Directory	Directory packet

(continued)

Table 2–229. Valid ?XFSTAT File Types

File Type	Meaning	Output Packet Format
?FDIR	Disk Directory	Directory packet
?FDKU	Disk unit	Unit packet
?FDPB	Business BASIC data base file	Other packet
?FFCC	FORTRAN Carriage Control	Other packet
?FGFN	Generic filename	Other packet
?FGLT	Generic Labeled Tape	IPC packet
?FIPC	IPC Port Entry	IPC packet
?FLCC	FORTRAN Carriage Control	Other packet
?FLDU	Logical disk unit	Directory packet
?FLNK	Link File	Other packet
?FLOG	System log file	Other packet
?FLPD	Data channel line printer 2	Unit packet
?FLPE	Data channel line printer (laser)	Unit packet
?FLPU	Data channel line printer	Unit packet
?FLUG	System Network Architecture Logical Unit Group	Other packet
?FMCU	Multiprocessor communications unit	Unit packet
?FMTF	Magnetic tape file	Other packet
?FNCC	FORTRAN Carriage Control	Other packet
?FOCC	FORTRAN Carriage Control	Other packet
?FPGR	AOS program file	Other packet
?FPIP	Pipe File	Other packet
?FPRG	AOS Program File	Other packet
?FPRV	AOS/VS Program File	Other packet
?FQUE	Queue Entry	IPC packet
?FSDF	System Data File	Other packet
?FSPR	Spoolable Peripheral Directory	IPC packet

(continued)

Table 2–229. Valid ?XFSTAT File Types

?FSTF	Symbol Table File	Other packet
?FSYN	Synchronous communication line	Other packet
?FTXT	Text File	Other packet
?FUDF	User Data File	Other packet
?FUNX	MV/UX File	Other packet
?FUPF	User Profile File	Other packet
?FVLF	Business BASIC volume load file	Other packet
?FWRD	Word processing file	Other packet

(concluded)

IPC Packet

Except for offset ?XFSTAT_PKT.GLOBAL_PORT, Figure 2–267 also shows the common set of input and output packet offsets. Table 2–230 describes the offsets. The offset ?XFSTAT_PKT.GLOBAL_PORT is unique to the IPC packet.

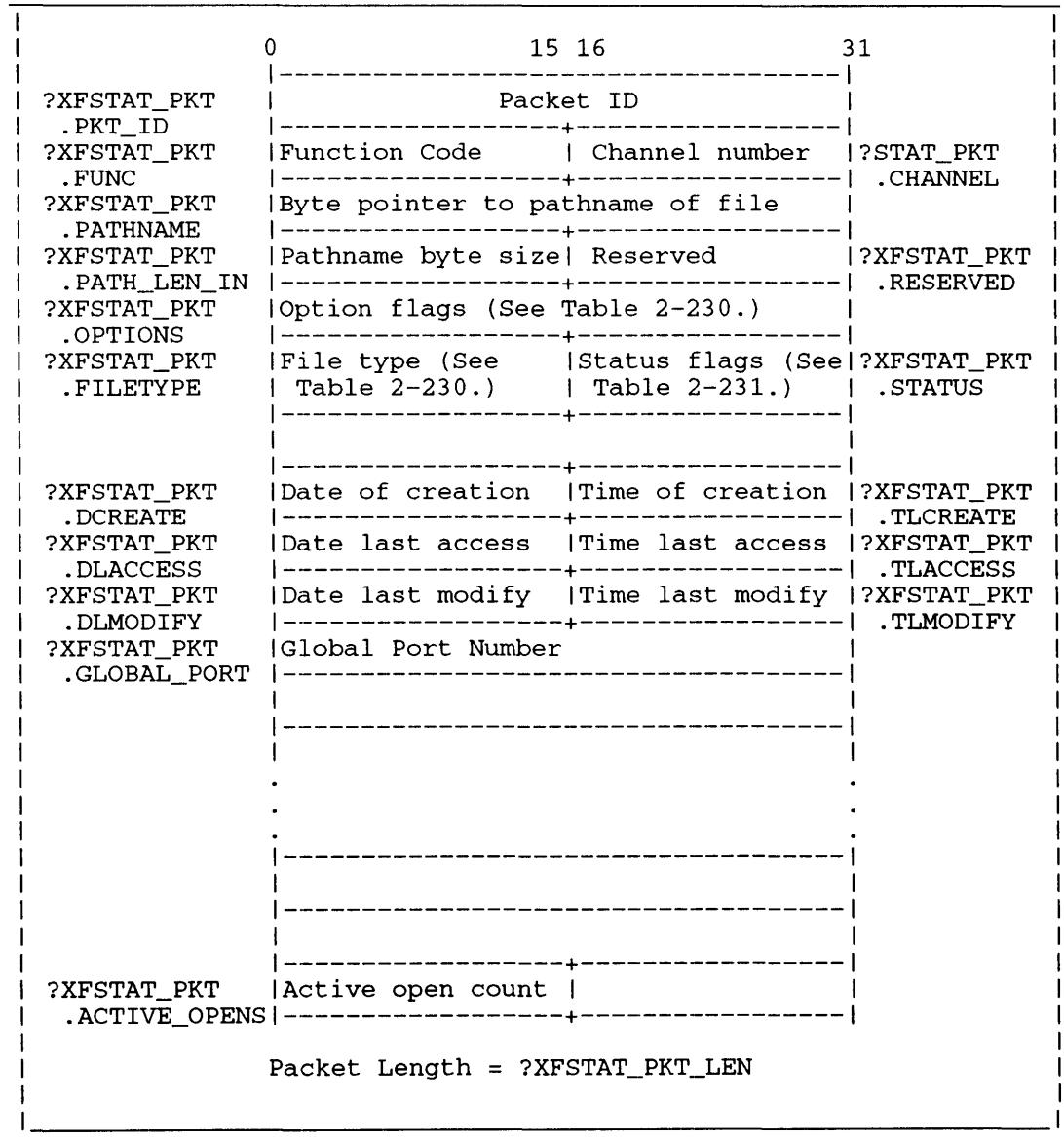


Figure 2-267. Structure of ?XFSTAT IPC Packet

Table 2-230. ?XFSTAT IPC File Status

Offset	Contents	Type
?XFSTAT_PKT .PKT_ID	Packet Identifier. (Set to ?XFSTAT_PKT_PKTID.)	In
?XFSTAT_PKT .FUNC	Function code; set to 0. ?XFSTAT does not use subpackets. The caller does not know in advance the type of file.	In
?XFSTAT_PKT .CHANNEL	Channel number of file. If you provide the channel number, you must also set the ?XFSTAT_OPTIONS_CHANNEL flag.	In
?XFSTAT_PKT .PATHNAME	Byte pointer to pathname of file. If you provide the byte pointer, you must also set the ?XFSTAT_OPTIONS_PATHNAME flag.	In
?XFSTAT_PKT .PATH_LEN_IN	Number of bytes in pathname, including null delimiter. You must provide this value if you specify a byte pointer.	In
?XFSTAT_PKT .RESERVED	Reserved. (Set to 0.)	In
?XFSTAT_PKT .OPTIONS	Option flags. Can be any of the following: ?XFSTAT_OPTIONS_NO_LINK_RESOLVE Set this flag when you do not want the file system to resolve links in the pathname. ?XFSTAT returns the status on the first detected link in the pathname. If the flag is not set and the pathname ends in a link, the call resolves the link entry and returns status information about the file to which the link refers. ?XFSTAT_OPTIONS_CHANNEL Set this flag when you specify the channel number. ?XFSTAT_OPTIONS_PATHNAME Set this flag when you specify the pathname.	In

(continued)

Table 2–230. ?XFSTAT IPC File Status

Offset	Contents	Type
	?XFSTAT_OPTIONS_GET_DISK_INFO Set this flag when you want the file system to retrieve additional disk unit information. If the flag remains set for all of the file types, the file system ignores the flag except for a disk unit status. The file system returns the additional disk unit information as zero values when you do not set the flag and the file resolves to a disk unit.	
?XFSTAT_PKT .FILETYPE	File type. Table 2-229 lists the file types and the output packet format.	Out
?XFSTAT_PKT .STATUS	Characteristics of file. (See Table 2-231.)	Out
?XFSTAT_PKT .DCREATE	Date of file creation. The call returns a date and time as single-precision integers. A date is expressed in days from December 31, 1967 to the date the file was create, access or modified.	Out
?XFSTAT_PKT .TCREATE	Time of file creation. A time is expressed as the number of biseconds (half the number of seconds) since midnight.	Out
?XFSTAT_PKT .DLACCESS	Date of last access to file.	Out
?XFSTAT_PKT .TLACCESS	Time of last access to file.	Out
?XFSTAT_PKT .DLMODIFY	Date of last modification to file.	Out
?XFSTAT_PKT .TLMODIFY	Time of last modification to file.	Out
?XFSTAT_PKT .GLOBAL_PORT	IPC Global Port Number.	In
?XFSTAT_PKT .ACTIVE_OPENS	Number of active open requests on the file. (The number of ?OPEN calls without comparable ?CLOSE calls.)	Out

(concluded)

Table 2-231. ?XFSTAT Status Flags

Flag Symbol	Flag Description
?XFSTAT_STATUS_MODIFIED	File was modified.
?XFSTAT_STATUS_PERMANENT	File/directory has permanence attribute.
?XFSTAT_STATUS_DEL_LAST_CLOSE	File is deleted after the last ?CLOSE.
?XFSTAT_STATUS_UDA	File has an associated UDA (user data area).
?XFSTAT_STATUS_UNIV_OWNER	Universal Owner access to the file/directory.
?XFSTAT_STATUS_UNIV_WRITE	Universal Write access to the file/directory.
?XFSTAT_STATUS_UNIV_APPEND	Universal Append access to the file/directory.
?XFSTAT_STATUS_UNIV_READ	Universal Read access to the file/directory.
?XFSTAT_STATUS_UNIV_EXEC	Universal Execute access to file/directory.

Directory Packet

Figure 2–268 shows the packet offsets for the directory packet. Table 2–232 describes the offsets returned by the operating system. Table 2–230 describes the common input offsets.

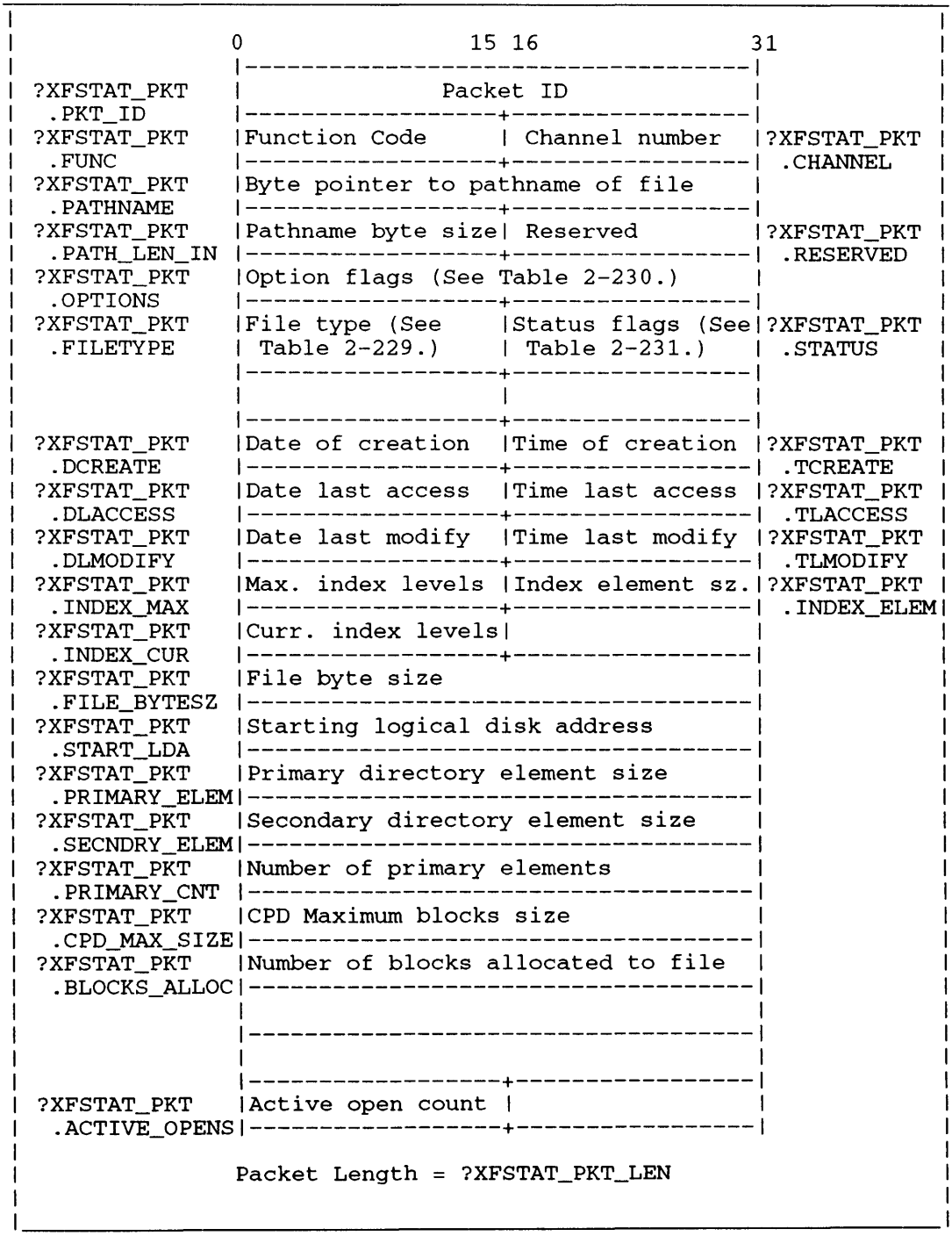


Figure 2–268. Structure of ?XFSTAT Directory Packet

Table 2–232. ?XFSTAT Directory and Other Packet File Status

Offset	Contents	Type
?XFSTAT_PKT .REC_FORM	Record format. Can be one of the following flags: ?XFSTAT_RCDFRM_DYNAMIC (Dynamic) ?XFSTAT_RCDFRM_DATA_SEN (Data-sensitive) ?XFSTAT_RCDFRM_FIX_LEN (Fixed-length) ?XFSTAT_RCDFRM_VAR_LEN (Variable-length) ?XFSTAT_RCDFRM_UNDEF_LEN (Undefined-length) ?XFSTAT_RCDFRM_BLK_REC_VAR (Variable block and record)	Other
?XFSTAT_PKT .REC_LEN	Maximum record length for fixed-length record files. The offset is undefined for all other types of files.	Other
?XFSTAT_PKT .INDEX_MAX	Maximum number of index levels.	Both
?XFSTAT_PKT .INDEX_ELEM	Index element size.	Both
?XFSTAT_PKT .INDEX_CUR	Current number of index levels. This value should not exceed the value of ?XFSTAT_PKT.INDEX_MAX.	Both
?XFSTAT_PKT .HOST_ID	Host Identifier for RMA files (?FREM) only. (Assigned 0 for all other file types.)	Other
?XFSTAT_PKT .FILE_BYTESZ	Byte size of file to the end of the file. You can only retrieve the length of a file after you have opened, reopened, or just created the file. You cannot retrieve the length correctly while you are currently writing to the file.	Both

(continued)

Table 2–232. ?XFSTAT Directory and Other Packet File Status

0	15 16	31	
?XFSTAT_PKT .PKT_ID	Packet ID		
?XFSTAT_PKT .FUNC	Function Code	Channel number	?XFSTAT_PKT .CHANNEL
?XFSTAT_PKT .PATHNAME	Byte pointer to pathname of file		
?XFSTAT_PKT .PATH_LEN_IN	Pathname byte size	Reserved	?XFSTAT_PKT .RESERVED
?XFSTAT_PKT .OPTIONS	Option flags (See Table 2-230.)		
?XFSTAT_PKT .FILETYPE	File type (See Table 2-229.)	Status flags (See Table 2-231.)	?XFSTAT_PKT .STATUS
?STAT_PKT .DCREATE	Date of creation	Time of creation	?XFSTAT_PKT .TCREATE
?XFSTAT_PKT .DLACCESS	Date last access	Time last access	?XFSTAT_PKT .TLACCESS
?XFSTAT_PKT .DLMODIFY	Date last modify	Time last modify	?XFSTAT_PKT .TLMODIFY
?XFSTAT_PKT .IOC_OR_CHAN	IOC/Channel number	Device code/Node	?XFSTAT_PKT .DEV_OR_NODE
?XFSTAT_PKT .UNIT_NUM	Unit number		
?XFSTAT_PKT .BUS_TYPE	Bus type	Device type	?XFSTAT_PKT .DEVICE_TYPE
?XFSTAT_PKT .DRIVER_TYPE	Driver type	Tape Def. Density	?XFSTAT_PKT .DEF_DENSITY
?XFSTAT_PKT .TRANSFER_SIZE	Tape transfer byte size		
?XFSTAT_PKT .DSK_SMR_SECTOR	Disk Second Most Reliable Sector		
?XFSTAT_PKT .DSK_SECTORS	Disk sectors total		
?XFSTAT_PKT .DSK_PATTERN1	Disk Pattern 1	Disk Pattern 2	?XFSTAT_PKT .DSK_PATTERN2
?XFSTAT_PKT .DSK_PATTERN3	Disk Pattern 3	Disk Pattern 4	?XFSTAT_PKT .DSK_PATTERN4
?XFSTAT_PKT .DSK_PATTERN5	Disk Pattern 5	Disk Pattern 6	?XFSTAT_PKT .DSK_PATTERN6
?XFSTAT_PKT .DSK_PATTERN7	Disk Pattern 7	Disk Pattern 8	?XFSTAT_PKT .DSK_PATTERN8
?XFSTAT_PKT .ACTIVE_OPENS	Active open count	Pattern count	?XFSTAT_PKT .DSK_PATTERN_CNT
	Packet Length = ?XFSTAT_PKT_LEN		

Other Packet

Figure 2–269 shows the packet offsets for the other packet. Table 2–232 describes the offsets returned (output) by the operating system. Table 2–230 describes the common input offsets.

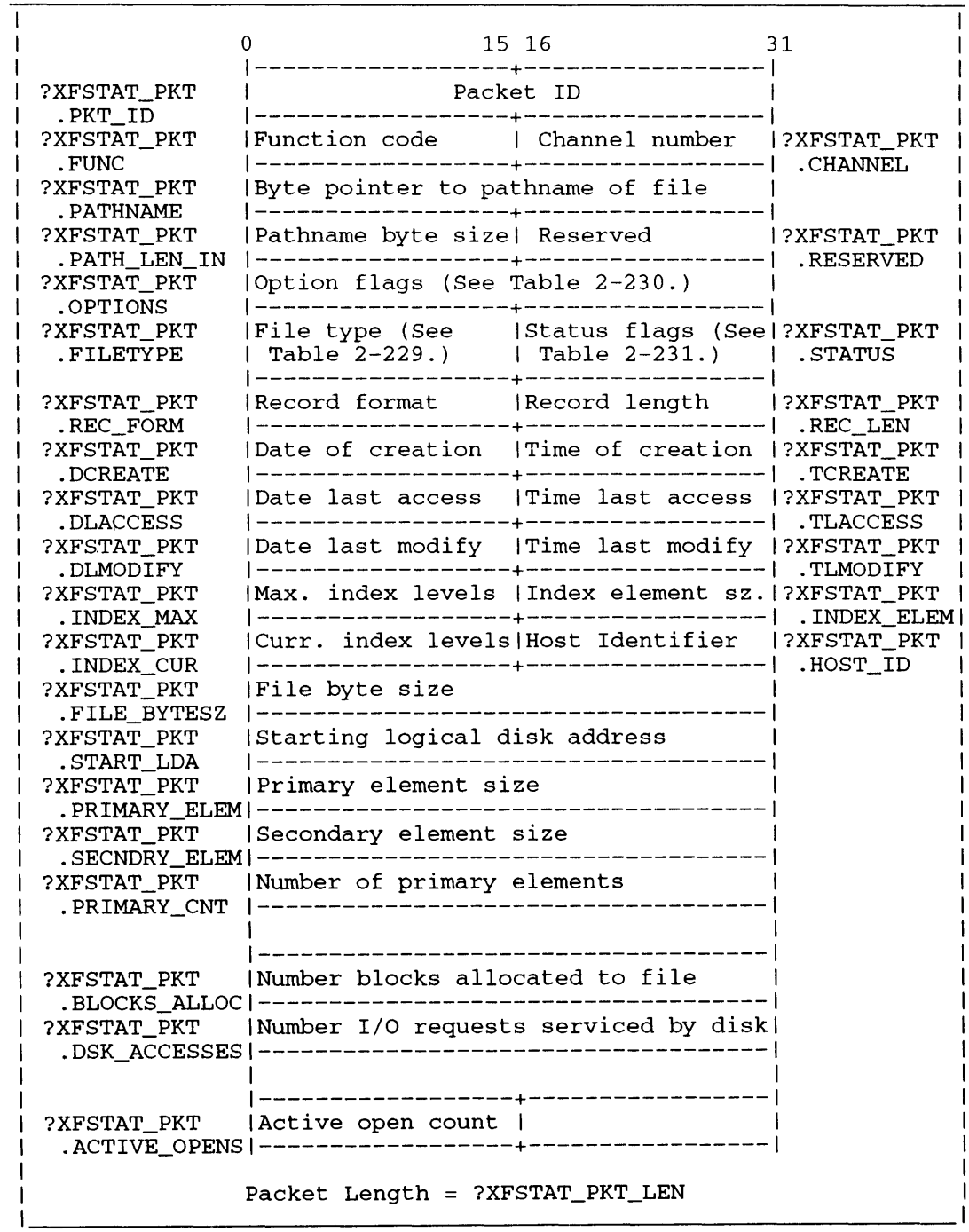


Figure 2–269. Structure of ?XFSTAT Other Packet

Unit Packet

Figure 2–270 shows the packet offsets for the unit packet. Table 2–230 describes the common input and output offsets. Table 2–233 describes the Unit Packet offsets returned (output) by the operating system.

The call returns additional disk unit file information (?XFSTAT_PKT.DSK_SMR_SECTOR through ?XFSTAT_PKT.DSK_PATTERN8 and ?XFSTAT_PKT.DSK_PATTERN_CNT) when you set the ?XFSTAT_OPTIONS_GET_DISK_INFO option flag and the file resolves to a disk unit. The information is based on hardware characteristics obtained from the disk. The request can be fulfilled, only when the physical unit is on line and the unit is not currently exclusively opened by another process. Otherwise, a physical–unit–off–line or disk–unit–opened–exclusively error occurs.

Device units may be connected to either an ECLIPSE or MRC bus. You must examine the bus–type field, ?XFSTAT_PKT.BUS_TYPE, to interpret the device address correctly. An ECLIPSE I/O device address is composed of an ECLIPSE I/O Channel (IOC) number, ECLIPSE I/O device code, and a unit number. An MRC device address is composed of an MRC channel number, MRC node number, and unit number.

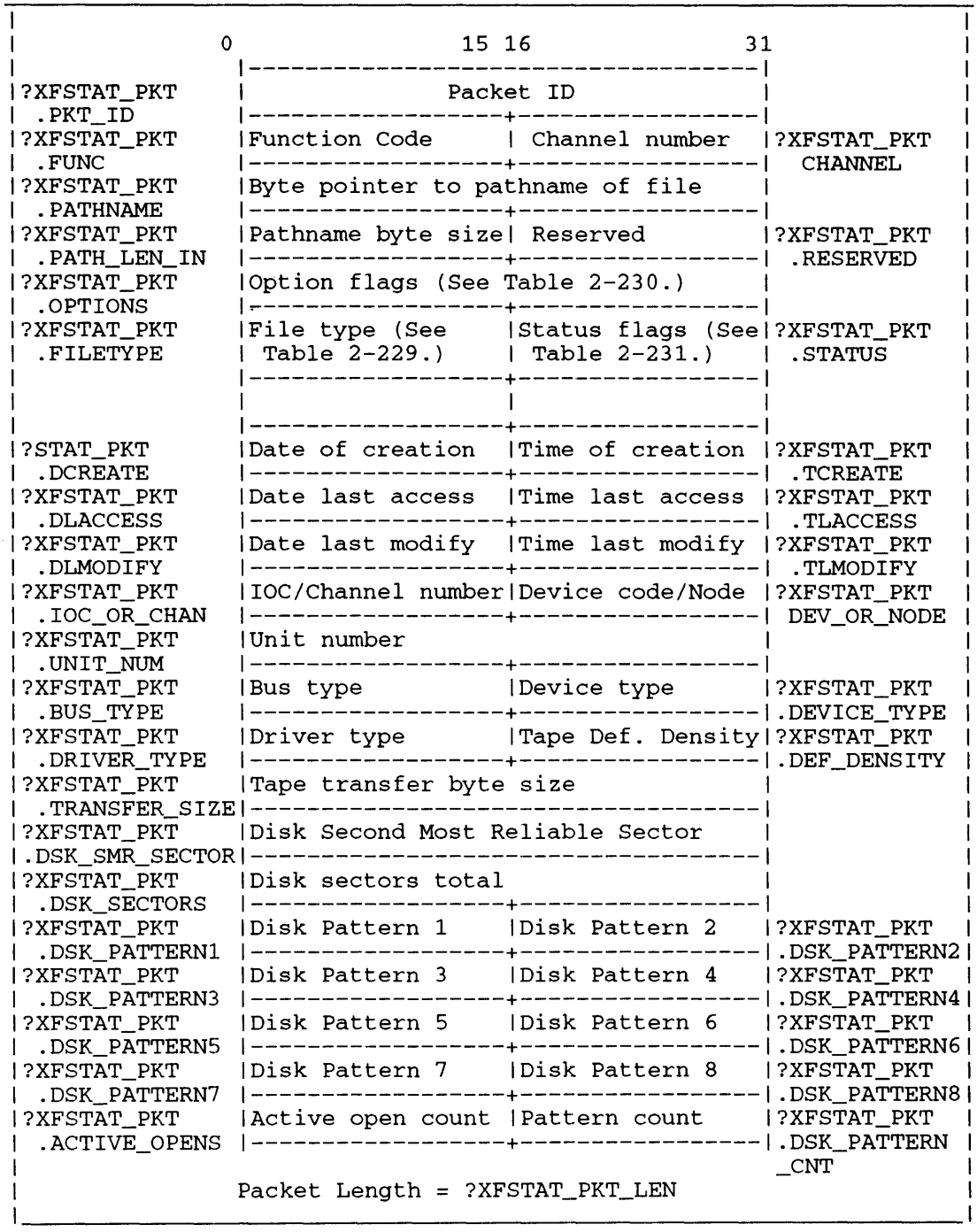


Figure 2–270. Structure of ?XFSTAT Unit Packet

Table 2–233. ?XFSTAT Unit Packet File Status

Offset	Contents
?XFSTAT_PKT .IOC_OR_CHAN	ECLIPSE IOC or MRC channel number.
?XFSTAT_PKT .DEV_OR_NODE	ECLIPSE I/O device code or MRC node number.
?XFSTAT_PKT .UNIT_NUM	ECLIPSE I/O or MRC unit number.
?XFSTAT_PKT .BUS_TYPE	Defines the bus type that connects to the device and can be one of the following: ?XFSTAT_ECLIPSE_BUS Device address pertains to ECLIPSE bus. ?XFSTAT_MRC_BUS Device address pertains to MRC bus.
?XFSTAT_PKT .DEVICE_TYPE	Type of ECLIPSE I/O or MRC device unit and can be one of the following: ?XFSTAT_DEVICE_DISK ?XFSTAT_DEVICE_TAPE ?XFSTAT_DEVICE_PRINTER ?XFSTAT_DEVICE_MCA
?XFSTAT_PKT .DEVICE_TYPE	With an ECLIPSE I/O driver, the response is one of the following: ?XFSTAT_DRIVER_DPF disk ?XFSTAT_DRIVER_DPJ disk ?XFSTAT_DRIVER_DPM disk ?XFSTAT_DRIVER_DPI disk ?XFSTAT_DRIVER_MTB tape ?XFSTAT_DRIVER_MTC tape ?XFSTAT_DRIVER_MTD tape ?XFSTAT_DRIVER_MTJ tape ?XFSTAT_DRIVER_LPB printer ?XFSTAT_DRIVER_LPD printer ?XFSTAT_DRIVER_LPE printer ?XFSTAT_DRIVER_LPJ printer ?XFSTAT_DRIVER_MCAT MCA transmitter ?XFSTAT_DRIVER_MCAR MCA receiver With an MRC driver, the response is one of the following: ?XFSTAT_DRIVER_NOT_APPLICABLE None defined. MRC has no driver type.

(continued)

Table 2–233. ?XFSTAT Unit Packet File Status

Offset	Contents
?XFSTAT_PKT .DEF_DENSITY	Returns default density (bytes per inch) of the tape unit. It can be one of the following: ?XFSTAT_DENSITY_800 800 BPI density ?XFSTAT_DENSITY_1600 1600 BPI density ?XFSTAT_DENSITY_6250 6250 BPI density ?XFSTAT_DENSITY_AUTO Automatic density ?XFSTAT_DENSITY_LOW Low density ?XFSTAT_DENSITY_MED Medium density ?XFSTAT_DENSITY_HIGH High density ?XFSTAT_DENSITY_NC No change
?XFSTAT_PKT .TRANSFER_SIZE	Maximum tape transfer buffer byte size.
?XFSTAT_PKT .DSK_SMR _SECTOR	Second most reliable (SMR) sector. The file system uses the SMR for each disk and stores a copy of the physical disk information table in the sector. The SMR sector location varies according to the disk type and size.
?XFSTAT_PKT .DSK_SECTORS	Total number of unformatted disk blocks available.
?XFSTAT_PKT .DSK_PATTERN1	First disk test pattern.
?XFSTAT_PKT .DSK_PATTERN2	Second disk test pattern.
?XFSTAT_PKT .DSK_PATTERN3	Third disk test pattern.
?XFSTAT_PKT .DSK_PATTERN4	Fourth disk test pattern.
?XFSTAT_PKT .DSK_PATTERN5	Fifth disk test pattern.
?XFSTAT_PKT .DSK_PATTERN6	Sixth disk test pattern.
?XFSTAT_PKT .DSK_PATTERN7	Seventh disk test pattern.
?XFSTAT_PKT .DSK_PATTERN8	Eighth disk test pattern.
?XFSTAT_PKT .DSK_PATTERN _CNT	Number of valid disk test patterns returned in the packet. The patterns test for bad blocks prior to disk formattting.

(concluded)

Disk test patterns refer to the bad block verification that is usually performed on disks prior to formatting. Error detection programs are run against the patterns written to the disk to find the bad blocks.

The intensity of error checking is proportional to the number of patterns used. The number and type of patterns may differ between and within disk classes. There can exist up to eight distinct pattern fields. A pattern count field identifies the number of valid pattern fields, which are sequentially ordered from patrn one. Unused fields are set to zero.

16–bit System Call Support

The ?XFSTAT system call supports a 16–bit version of the call. However, 16–bit callers are responsible for supplying the packet address in the 16–bit address format, which includes the ring field. Callers will use the same packet offset symbols as defined for 32–bit call.

Example with Pathname Supplied

The caller wants file information for the entry specified in the target pathname, :UDD:BRIANNE:MEMO.LS. In the ?XFSTAT packet, the caller should set the packet as follows:

?XFSTAT_PKT.PKT_ID	= ?XFSTAT_PKT_PKTID
?XFSTAT_PKT.FUNC	= 0
?XFSTAT_PKT.CHANNEL	= 0
?XFSTAT_PKT.PATHNAME	= byte pointer to :UDD:BRIANNE:MEMO.LS
?XFSTAT_PKT.PATH_LEN_IN	= 22 (Caller's pathname buffer is 22 bytes, including null delimiter)
?XFSTAT_PKT.RESERVED	= 0
?XFSTAT_PKT.OPTIONS	= ?XFSTAT_OPTIONS_PATHNAME

Sets AC2 = word addr. (?XFSTAT_PKT).

After issuing the call, the packet's output fields contain the requested information. The input fields remain unchanged. All packet values are in octal.

The packet information tells the caller that file, MEMO.LS, is a user data file (?FUDF); is currently a 1 index level file; has a byte size of 101144; has 100 blocks allocated; its primary element size is 104 while its secondary element size is 4. Determining the values in parentheses are normally found by the symbolic flags defined.

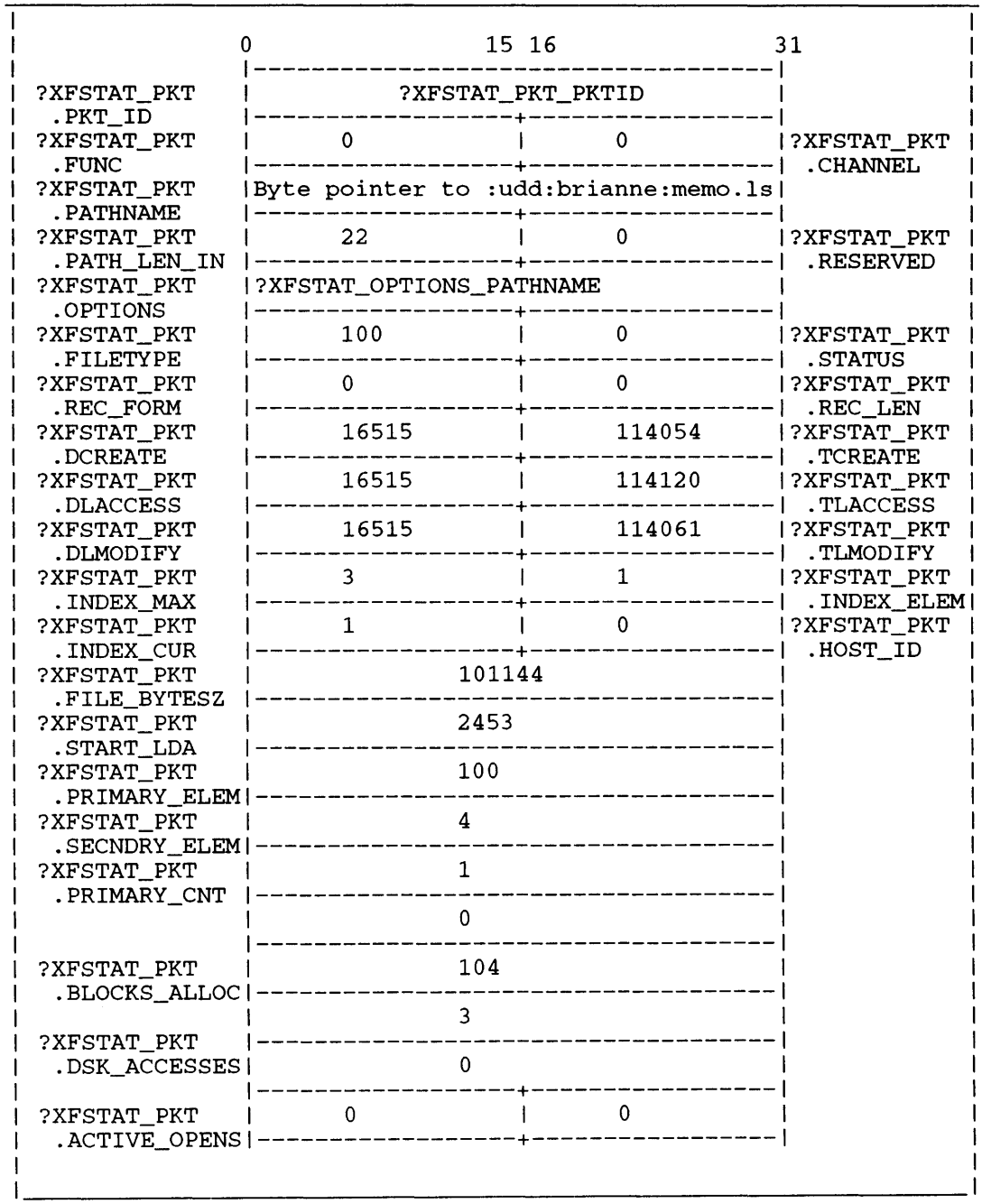


Figure 2-271. Example of Pathname Supplied

Example with Channel Number Supplied

The caller has a channel number to a directory, :UDD:BRIANNE, The caller previously opened it and wants the file status on it. In the ?XFSTAT packet, the caller should set the packet as follows:

```
?XFSTAT_PKT.PKT_ID          = ?XFSTAT_PKT_PKTID
?XFSTAT_PKT.FUNC            = 0
?XFSTAT_PKT.CHANNEL         = 32
?XFSTAT_PKT.PATHNAME        = 0
?XFSTAT_PKT.PATH_LEN_IN     = 0
?XFSTAT_PKT.RESERVED        = 0
?XFSTAT_PKT.OPTIONS         = ?XFSTAT_OPTIONS_CHANNEL
```

Sets AC2 = word addr. (?XFSTAT_PKT).

After issuing the call, the packet's output fields contain the requested information. The input fields remain unchanged. All packet values are in octal. The packet information tells the caller that the file

- is a directory file type (?FDIR) with currently a 1 index level directory
- has an index element size of 2
- uses a byte size of 1000
- uses a starting logical disk address of 10751
- has a primary directory element size of 2 blocks
- has an allowance of 2 primary directory elements to be allocated before going to the secondary element
- uses a 1 block secondary directory element size
- has 1222 disk blocks currently allocated within it
- has an open count of 1

The status flags word indicates this file has the permanence attribute on and universal Read and Execute access rights.

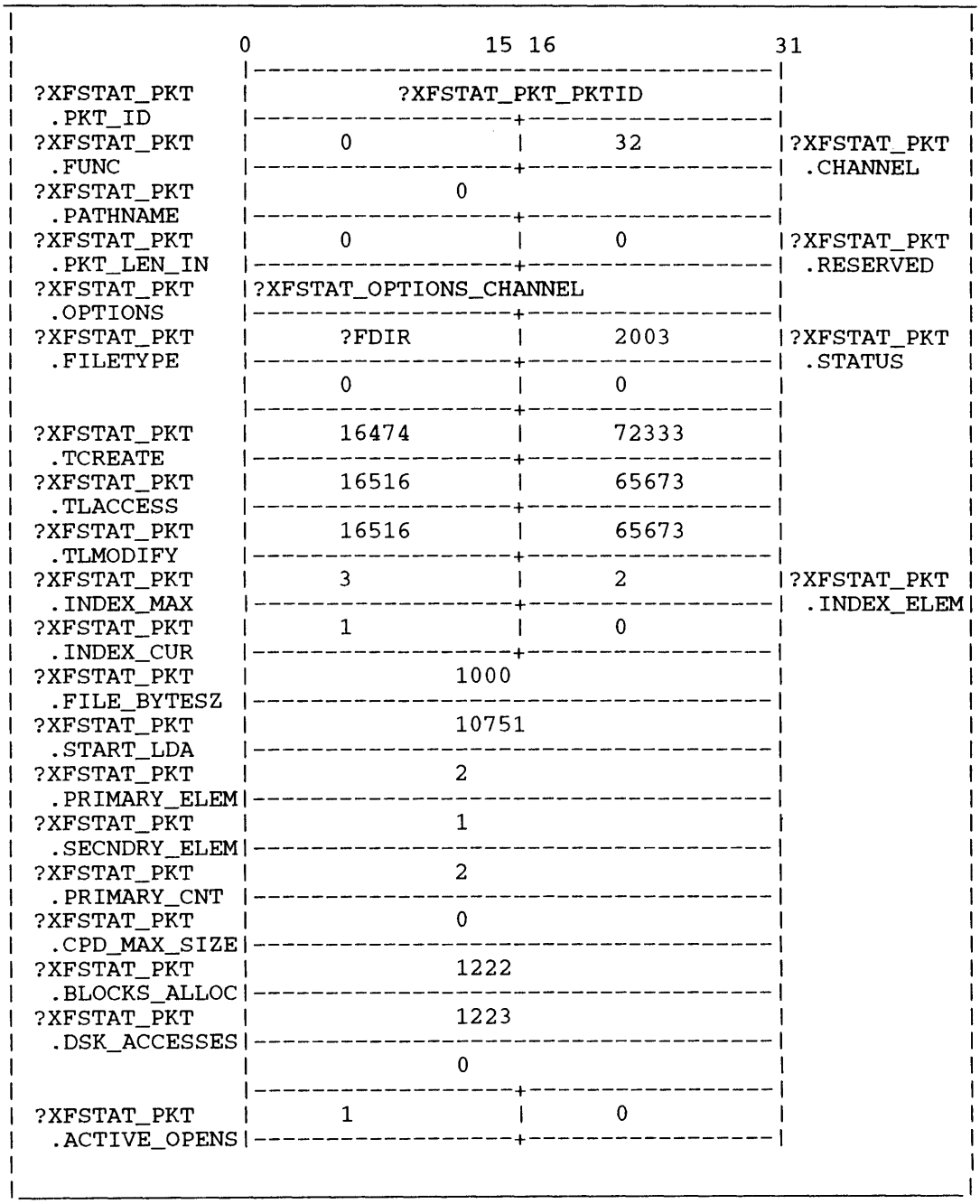


Figure 2-272. Example of Directory Type Grouping

Notes

- Should additional information be included in the future, DG will assign a new packet identifier. To access the new information, the caller should specify the most recent packet identifier and packet length constant.
- When you set the ?XFSTAT_OPTIONS_GET_DISK_INFO flag and the file type resolves to a disk unit, the ?XFSTAT call requires additional time to open the disk (if necessary) and to read the information. If the flag remains set for all of the file types, the file system ignores the flag except for a disk unit status.

AOS/VS II only

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

Input

AC0	Reserved (Set to 0)
AC1	Reserved (Set to 0)
AC2	?XGTACP Packet Address unless specified as a system call parameter.

Output

AC0	Error code
AC1	Unchanged
AC2	Unchanged

Error Codes in AC0

ERFNO	Channel not open
ERICN	Illegal channel
ERPKT	Invalid packet ID
ERPRE	Invalid system call parameter
ERPRH	Attempt to access process not in hierarchy
ERPRV	Caller not privileged for this action
ERRVN	Reserved value not zero
ERVBP	Invalid byte pointer passed as a system call argument
ER_FS_ILLEGAL_GROUP_LIST_FORMAT	
ER_FS_GROUPNAME_TOO_LONG	
ER_FS_ILLEGAL_GROUPNAME_CHARACTER	
ER_FS_INVALID_GROUP_LIST_BYTE_PTR	
ER_FS_INVALID_PATHNAME_BYTE_PTR	
ER_FS_INVALID_PKT_PTR	
ER_FS_INVALID_USERNAME_BYTE_PTR	
ER_FS_INVALID_XGTACP_FUNCTION	
ER_FS_TOO_MANY_GROUPS_SPECIFIED	
ER_FS_USERNAME_TOO_LONG	
ER_FS_ZERO_LENGTH_USERNAME	
ER_FS_DIRECTORY_NOT_AVAILABLE	Directory not available because the LDU was force released (AOS/VS II only)

Why Use It?

?XGTACP returns the access privileges for a specific file and username/group list combination. You can issue ?XGTACP 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 the directory of the file to determine your own access rights to the file. When you set ?XGTACP_USER_OR_PID to –1, the call returns the access rights based on the current username/group list combination of the calling process. Others must have the Superuser privilege.

What It Does

?XGTACP returns the access privileges of a file when given a username, an optional list of groups, and a filename (or channel number). The algorithm for determining access is the same as it is for normal access control. It is a first match algorithm in which the order of the terms in the ACL can determine what access rights are returned.

For example, if an ACL reads WARNER:GROUP1,RE WARNER:GROUP2,OWARE, Warner in the GROUP1 group gets RE access no matter to what other groups Warner may be belong, including GROUP2. Warner only gets OWARE access if the username WARNER is in GROUP2 and not GROUP1. The first term in the ACL (which is parsed from left to right) matches the username and one of the groups in the group access control list. The first term would have to fail the match in order for the second term to even be examined.

If a user supplies a zero in the group access control list field, then no groups are used. The only match would occur with an ACL term that has no groups. For example, if the ACL reads WARNER:GROUP1,RE WARNER:GROUP2,WRE WARNER,OWARE and if ?XGTACP is issued with a username of WARNER and no group access control list, OWARE access is returned. The first two terms (WARNER:GROUP1 and WARNER:GROUP2) do not match.

If the user supplies a PID number and filename (or channel number), the call returns the access that a particular process (identified by the PID number) has to the specified file based on the target process's username and current group access control list.

Figure 2–273 contains the structure of the ?XGTACP packet and Table 2–234 describes its contents.

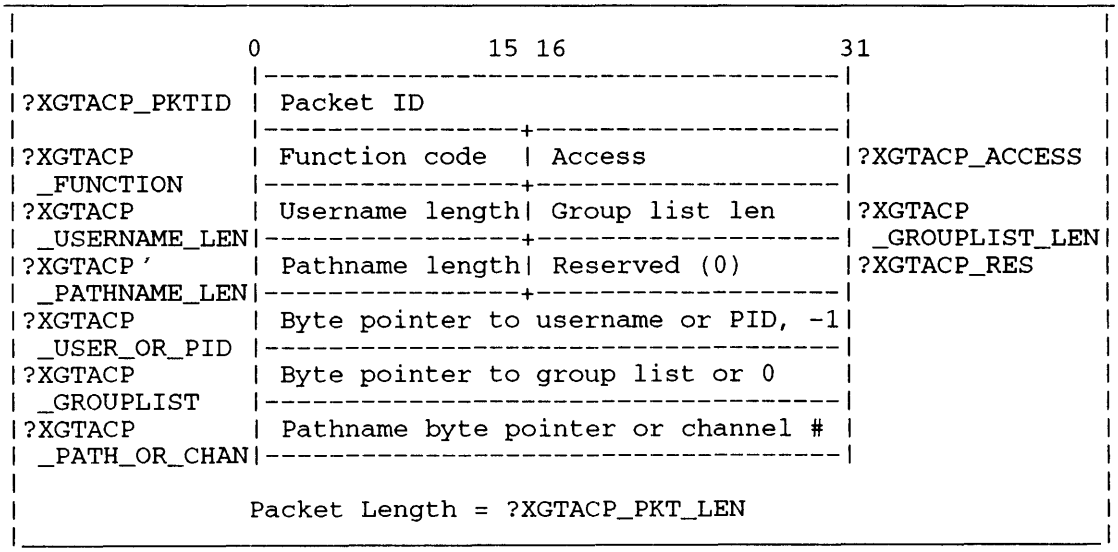


Figure 2–273. Structure of ?XGTACP Packet

Table 2–234. Contents of ?XGTACP Packet

Offset	Contents
?XGTACP_PKTID	Packet identifier. Place ?XGTACP_PKT_PKTID here.
?XGTACP_FUNCTION	The values for the offset can range from ?XGTACP_MIN_FUNC to ?XGTACP_MAX_FUNC. The values are ?XGTACP_USER_AND_PATH This function gets the access rights based on username, group access control list, and pathname specified by the caller. ?XGTACP_USER_AND_CHAN This function gets the access rights based on the username, group access control list, and channel number of the open file specified by the caller. ?XGTACP_PID_AND_PATH This function gets the access rights based on PID number of the process to verify and the pathname specified by the caller. ?XGTACP_PID_AND_CHAN This function gets the access rights based on the PID number of the process to verify and the channel number of the open file specified by the caller.
?XGTACP_ACCESS	The access rights returned are in the same format as ?GTACP (although ?GTACP returns them in an AC instead of a packet).
?XGTACP_USERNAME_LEN	The number of bytes in the username string.
?XGTACP_GROUPLIST_LEN	The number of bytes in the group access control list string.
?XGTACP_PATHNAME_LEN	The number of bytes in the pathname string.
?XGTACP_USER_OR_PID	Contains -1, PID, or the byte pointer to a username.
?XGTACP_GROUPLIST	Contains 0 or a byte pointer to the group access control list -- a double null-terminated list of groups. Group is specified as a null-terminated, 16-byte (including the null) ASCII string.
?XGTACP_PATH_OR_CHAN	The offset contains a channel number or a byte pointer to the pathname -- a null-terminated, 255-byte (including the null) ASCII string. Filename entries are separated by colons.

16–Bit System Call Support

The 16–bit version of ?XGTACP uses the same packet offsets as the 32–bit call. The 16–bit version also requires a ring field with the 16–bit address.

Notes

- See the description of the ?XPSTAT system call, which returns the group access control list of a process.

AOS/VS

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

Input

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

Output

AC0	Unchanged
AC1	Unchanged
AC2	Address of the ?XINIT packet

Error Codes in AC0

ER_FS_LDM_DISK_CONTAINS_MULTIPLE_LD_PIECES
: Disk contains multiple LD (logical disk) pieces

ERCNM
: Controller does not support LDU (logical disk unit) mirroring

ERDMO
: Disk marked as owned by another system

ERDRS
: Device reserved by another port

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

ERIFT
: Illegal file type

ERILD
: Inconsistent LD

ERIML
: Incomplete mirrored LDU specified

ER_FS_INVALID_NUM_IMAGES
: Invalid number of images

ER_FS_INVALID_NUM_UNITS
: Invalid number of physical units

ER_FS_INVALID_XINIT_FUNCTION
: Invalid ?XINIT function

ER_FS_INVALID_XINIT_OPTIONS
: Invalid ?XINIT options

ERLFM
: LDU format mismatch — not a valid mirror

ERLMM
: LDU name mismatch — not a valid mirror

ERLSZ
: LDU size mismatch — not a valid mirror

ERMIS
: Disk and file system revision numbers don't match

ERMSY
: Mirrored LDU is not synchronized

ERNAE
: Filename already exists

ERNAM
: LDU is not mirrored

ER_FS_LDM_DISK_DOES_NOT_CONTAIN_ANY_LD_PIECES
: Disk contains no logical disk pieces

ER_FS_OWNER_ACCESS_REQ
: Owner access required (to LDU)

ER_FS_LDM_LD_PIECE_NOT_FOUND_ON_DISK
: Piece not found on disk

ER_FS_LDM_LDU_RELEASED_DURING_SYNC
: LDU released during synchronization

ERRID
: Nonunique logical disk unique IDs — not a valid mirror

ERSIP
: Cannot initialize an LDU image that was previously being synchronized

ERVBP
: Invalid byte pointer passed as a system call argument

ERVWP
: Invalid word pointer passed as a system call argument

ER_FS_WRITE_OR_APPEND_ACCESS_REQ
: Write or Append access required (to target directory)

ER_FS_CACHE_NOT_ENABLED
: Attempted to initialize an LDU for caching, but caching was not enabled at system startup

Why Use It?

You use ?XINIT (or ?INIT) to initialize a logical disk. In addition, you can use ?XINIT to itialize a set of synchronized LDU (logical disk unit) images.

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 either Write or Append access to the target directory. In addition, you must have Execute access to all the physical disks specified and to the target directory. ?XINIT's access requirements are the same ones that ?INIT must have.

What It Does

Initializing a set of synchronized LDU images via ?XINIT means that you intend to include at least one mirrored LDU into the existing file structure. If one LDU image becomes unavailable, you still have access to the LDU data on the remaining available image.

?XINIT sizes the disk units, creates memory databases, and recalibrates the disk.

?XINIT is an extended version of, and upward compatible with, ?INIT. We recommend that you use ?XINIT instead of ?INIT in all new programs that you write. There is no need for an extended release call since ?RELEASE has all the functionality required to handle a logical disk that either ?INIT or ?XINIT has initialized. Furthermore, you issue ?MIRROR to synchronize mirrored images that ?XINIT cannot initialize and to manage mirrored LDU configurations.

?XINIT allows you to initialize many types of LDUs. They may be mirrored and reside on multiple physical disks. Additionally, you may specify an LDU in the old or new file system syntax. The new file system syntax includes an LDUID — (LDU unique ID).

?XINIT Continued

The main packet contains pointers to subpackets that contain physical unit lists for each LDU image. You specify two images when you initlize a synchronized set of images (for mirrored LDUs). Figure 2–274 shows the structure of the ?XINIT main packet, and Table 2–235 describes its contents. Figure 2–275 shows the structure and contents of the ?XINIT subpacket. This subpacket is identical to the one that ?MIRROR uses except for the value of the packet identifier.

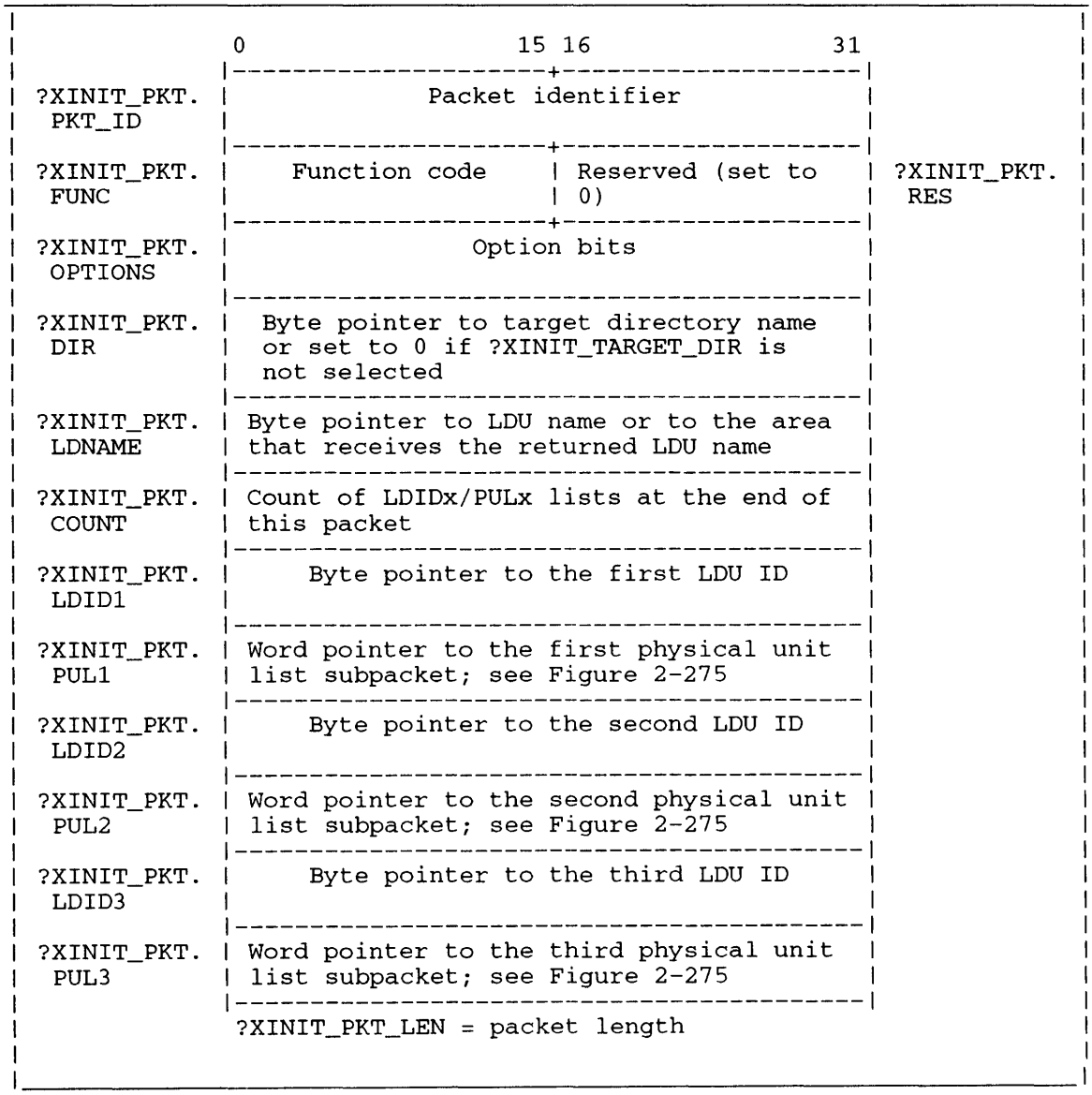


Figure 2–274. Structure of ?XINIT Packet

Table 2–235. Contents of ?XINIT Packet

Offset	Contents	
?XINIT_PKT.PKT_ID (doubleword)	Packet identifier. Place ?XINIT_PKT_PKTID here.	
?XINIT_PKT.FUNC	Function code. Place one of the following values here:	
	?XINIT_ROOT	To graft a LDU into the root directory (:).
	?XINIT_TARGET_DIR	To graft a LDU into the target directory that you specify in offset ?XINIT_PKT.DIR.
	?XINIT_WORKING_DIR	To graft a LDU into the working directory.
?XINIT_PKT.RES	Reserved. (Set to 0.)	
?XINIT_PKT.OPTIONS (doubleword)	Options word. Place one of the following values here:	
	?XINIT_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 ?XINIT_PKT.LDID1. This value applies only to the new file system.
	?XINIT_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.
	?XINIT_NO_HARDWARE	To prevent the system from mirroring the LDU in hardware; hardware mirroring is the default case whenever it is possible. This value applies only to the new file system.
	?XINIT_OVERRIDE	To override the error condition ERIML ("Incomplete mirrored LDU specified").
	?XINIT_CACHE	To initialize the LDU for caching

(continued)

Table 2–235. Contents of ?XINIT Packet

Offset	Contents
	The OS might return the following value here: ?XINIT_FIXUP_REC — To recommend that you run utility program FIXUP. This value applies only to the old file system.
?XINIT_PKT.DIR (doubleword)	If you specified option ?XINIT_TARGET_DIR in offset ?XINIT_PKT.FUNC, then offset ?XINIT_PKT.DIR must contain a byte pointer to the name of the directory into which the LDU will be grafted. In turn, this name must end with the null byte. If you didn't specify option ?XINIT_TARGET_DIR in offset ?XINIT_PKT.FUNC, then supply 0 here.
?XINIT_PKT.LDNAME (doubleword)	If all the physical disks contain only one logical disk, then supply a byte pointer here to an area ?MXFN bytes long. ?XINIT will return the LDU name in this area. Otherwise (the physical disks contain more than one logical disk piece), do the following steps: 1. Supply a byte pointer here to a null-terminated LDU name. 2. Set bit ?XINIT_LDID_SPECIFIED in offset ?XINIT_PKT.OPTIONS. 3. Supply, in offset ?XINIT_PKT.LDID1, a byte pointer to a null-terminated LD image name.
?XINIT_PKT.COUNT (doubleword)	Supply the nonzero number of LDIDx/PULx (where x is 1 or 2 for the old file system, 1 or 2 or 3 for the new file system) pairs of offsets that you supply next in this parameter packet. The number cannot exceed ?XINIT_MAXIMAGES, which is 2 for the old file system, and 3 for the new file system.
?XINIT_PKT.LDID1	If you have set bit ?XINIT_LDID_SPECIFIED in offset ?XINIT_PKT.OPTIONS, then supply a byte pointer to a valid and null-terminated LDU ID. If you have not set this bit, then supply 0. Always supply 0 for the old file system.
?XINIT_PKT.PUL1 (doubleword)	Supply a word pointer to the first physical unit list subpacket. Figure 2-275 describes this subpacket.

(continued)

Table 2-235. Contents of ?XINIT Packet

Offset	Contents
?XINIT_PKT.LDID2	If you have set bit ?XINIT_LDID_SPECIFIED in offset ?XINIT_PKT.OPTIONS and want to mirror, then supply a null-terminated byte pointer to a second valid LDU ID. If you have not set this bit -- or have set it and don't want to mirror -- then supply 0. Always supply 0 for the old file system.
?XINIT_PKT.PUL2 (doubleword)	Supply, if necessary (i.e., for mirroring), a word pointer to the second physical unit list subpacket. Figure 2-275 describes this subpacket. If you don't need the subpacket, supply 0.
?XINIT_PKT.LDID3	If you have set bit ?XINIT_LDID_SPECIFIED in offset ?XINIT_PKT.OPTIONS and want to mirror, then supply a null-terminated byte pointer to a third valid LDU ID. If you have not set this bit -- or have set it and don't want to mirror -- then supply 0. Always supply 0 for the old file system.
?XINIT_PKT.PUL3 (doubleword)	Supply, if necessary (i.e., for mirroring), a word pointer to the third physical unit list subpacket. Figure 2-275 describes this subpacket. If you don't need the subpacket, supply 0. Always supply 0 for the old file system.

(concluded)

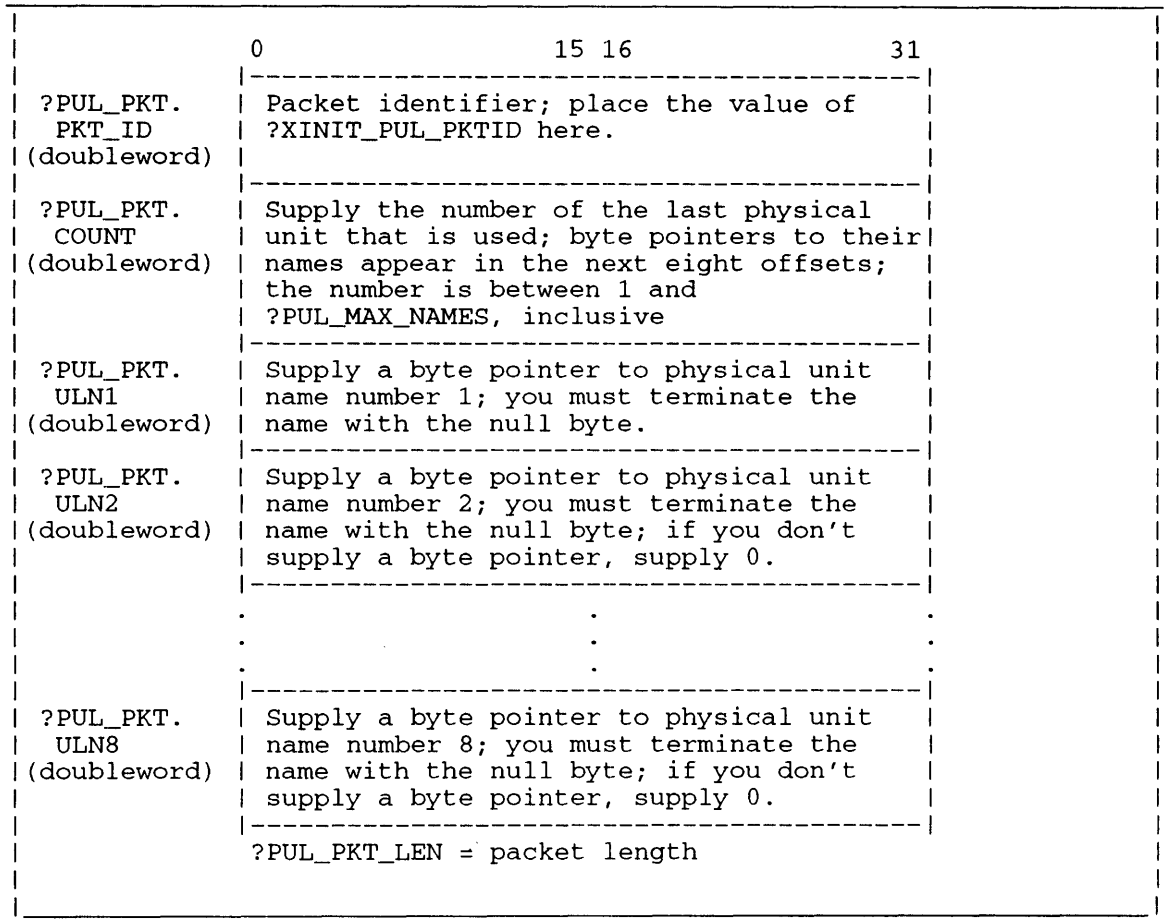


Figure 2–275. Structure of ?XINIT Subpacket

16–Bit System Call Support

A 16–bit version of this system call also exists. Its functionality is identical to that of the 32–bit version previously described.

Figure 2–276 contains the structure of the 16–bit ?XINIT call main packet and Figure 2–277 contains the structure of the 16–bit ?XINIT call subpacket. The definitions of the offsets in these packets are identical to those in Figure 2–274 and Figure 2–275, respectively. The only differences in the corresponding packets are the names of the offsets.

Table 2–236 describes the contents of Figure 2–276.

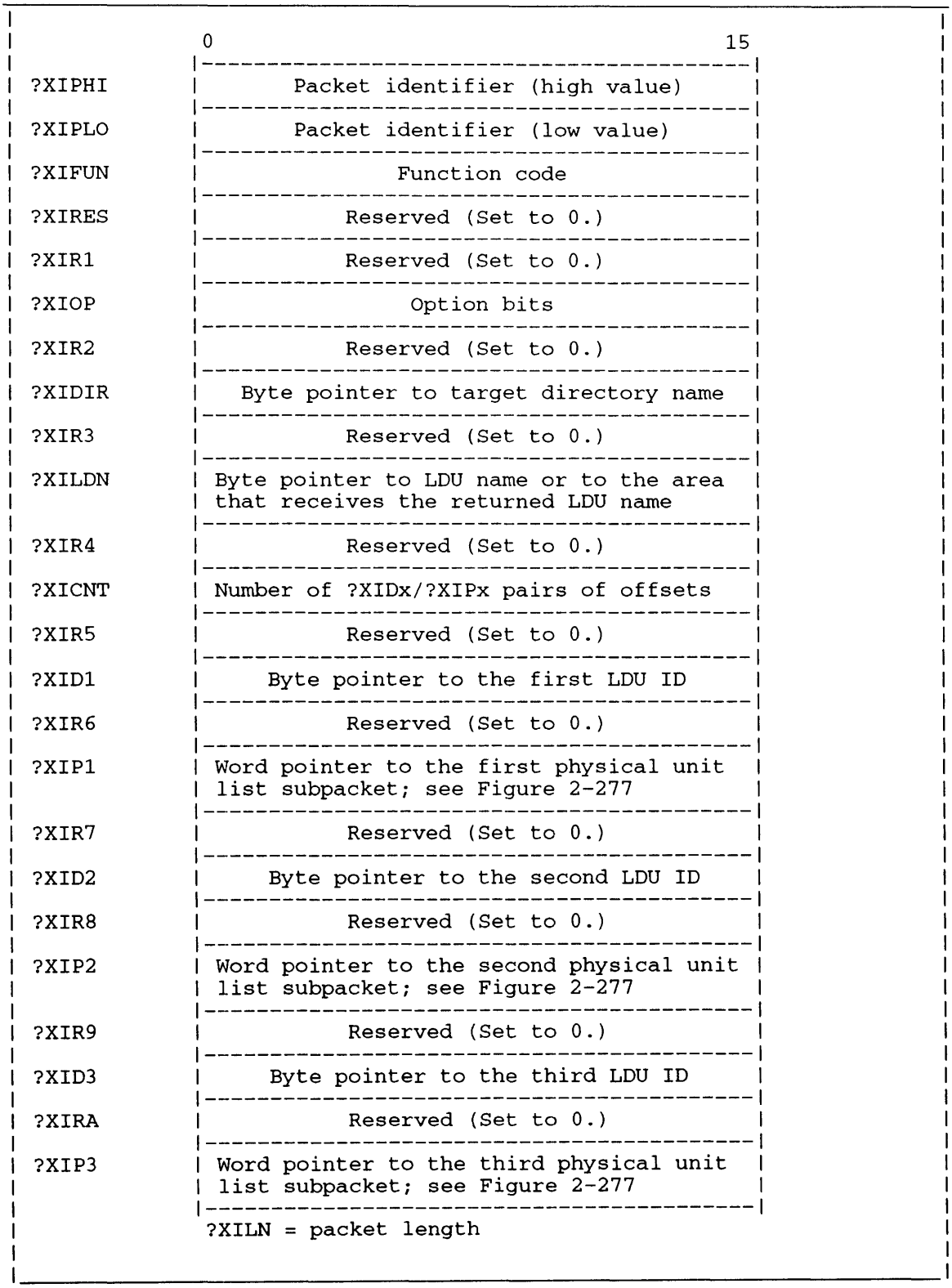


Figure 2–276. Structure of 16–Bit ?XINIT Packet

Table 2–236. Contents of 16–Bit ?XINIT Packet

Offset	Contents
?XIPHI	Packet identifier (high word). Place ?XII1 here.
?XIPLO	Packet identifier (low word). Place ?XII2 here.
?XIFUN	Function code. Place one of the following values here: ?XFGRT To graft a LDU into the root directory (:). ?XFGWD To graft a LDU into the working directory. ?XIGTD To graft a LDU into the target directory that you specify in offset ?XIDIR.
?XIRES	Reserved. (Set to 0.)
?XIR1	Reserved. (Set to 0.)
?XIOP	Options word. Place one of the following values here: ?XBIDS 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 ?XID1. This value applies only to the new file system. ?XBNHR To prevent the system from mirroring the LDU in hardware; hardware mirroring is the default case whenever it is possible. This value applies only to the new file system. ?XBTRP 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. ?XBORD To override the error condition ERIML ("Incomplete mirrored LDU specified"). The operating system might return the following value here: ?XBFXR To recommend that you run utility program FIXUP. This value applies only to the old file system.
?XIR2	Reserved. (Set to 0.)

(continued)

Table 2–236. Contents of 16–Bit ?XINIT Packet

Offset	Contents
?XIDIR	If you specified ?XIGTD in offset ?XIFUN, then offset ?XIDIR must contain a byte pointer to the name of the directory into which the LDU will be grafted. In turn, this name must end with a null byte. If you didn't specify option ?XIGTD in offset ?XIFUN, then supply 0 here.
?XIR3	Reserved. (Set to 0.)
?XILDN	If all the physical disks contain only one logical disk, then supply a byte pointer here to an area ?MXFN bytes long. ?XINIT will return the LDU name in this area. Otherwise (the physical disks contain more than one logical disk piece), do the following steps: 1. Supply a byte pointer here to a null-terminated LDU name. 2. Set bit ?XBIDS in offset ?XIOP. 3. Supply, in offset ?XID1, a byte pointer to a null-terminated LD image name.
?XIR4	Reserved. (Set to 0.)
?XICNT (doubleword)	Supply the nonzero number of ?XIDx/?XIPx (where x is 1 or 2 for the old file system, and 1 or 2 or 3 for the new file system) pairs of offsets that you supply next in this parameter packet. The number cannot exceed ?MAXIMAGES, which is 2 for the old file system, and 3 for the new file system.
?XIR5	Reserved. (Set to 0.)
?XID1	If you have set bit ?XBIDS in offset ?XIOP, then supply a byte pointer to a valid and null-terminated LDU ID. If you have not set this bit, then supply 0. Always supply 0 for the old file system.
?XIR6	Reserved. (Set to 0.)
?XIP1	Supply a word pointer to the first physical unit list subpacket. Figure 2-277 describes this subpacket.
?XIR7	Reserved. (Set to 0.)
?XID2	If you have set bit ?XBIDS in offset ?XIOP and want to mirror, then supply a null-terminated byte pointer to a second valid LDU ID. If you have not set this bit, or have set it and don't want to mirror, then supply 0. Always supply 0 for the old file system.
?XIR8	Reserved. (Set to 0.)

(continued)

Table 2–236. Contents of 16–Bit ?XINIT Packet

Offset	Contents
?XIP2	Supply, if necessary (i.e., for mirroring), a word pointer to the second physical unit list subpacket. Figure 2-277 describes this subpacket. If you don't need the subpacket, supply 0.
?XIR9	Reserved. (Set to 0.)
?XID3	If you have set bit ?XBIDS in offset ?XIOP and want to mirror, then supply a null-terminated byte pointer to a third valid LDU ID. If you have not set this bit, or have set it and don't want to mirror, then supply 0. Always supply 0 for the old file system.
?XIRA	Reserved. (Set to 0.)
?XIP3	Supply, if necessary (i.e., for mirroring), a word pointer to the third physical unit list subpacket. Figure 2-277 describes this subpacket. If you don't need the subpacket, supply 0.

(concluded)

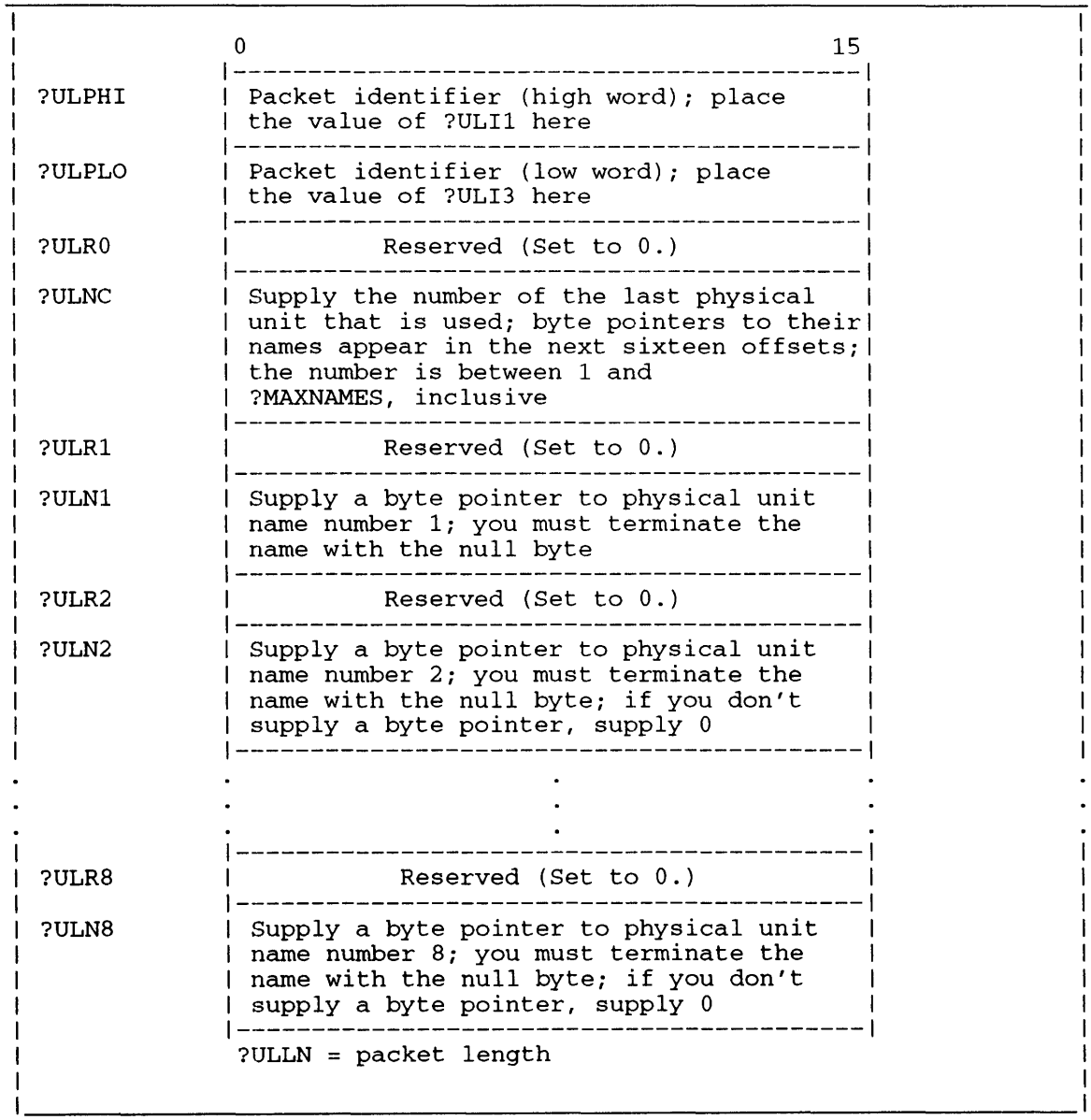


Figure 2–277. Structure of 16–Bit ?XINIT Subpacket

Notes

- See the descriptions of ?RELEASE, ?MIRROR, and ?XINIT in this chapter.

?XMT
error return
normal return

Input

AC0	Address of the mailbox that is to receive the intertask message
AC1	Intertask message
AC2	To broadcast the intertask message to all waiting, –1

Output

AC0	Unchanged
AC1	Unchanged
AC2	Unchanged

Error Ces in AC0

ERVWP	Invalid word pointer passed as a system call argument
ERXMT	Signal to address already in use (You tried to send a message to a mailbox that already contains a message.)
ERXMZ	Attempt to XMT illegal message

Why Use It?

?XMT is one of four system calls for intertask communications. (The others are ?XMTW, ?REC, and ?RECNW.) You can use the intertask communications facility to pass variables and data or to synchronize 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

?XMT sends a nonzero message to the mailbox that you specify in AC0. The mailbox must contain 0 before you issue ?XMT.

Before you issue ?XMT, load AC0 with the address of the mailbox, and load AC1 with the message. The receiving task gets the message by issuing either a ?REC or a ?RECNW against the same mailbox. (For 32–bit processes, the mailboxes are 32 bits wide; for 16–bit processes, the mailboxes are 16 bits wide.)

The broadcast option allows you to send the message to all other tasks that issue corresponding ?REC or ?RECNW system calls. If you do not broadcast the message and more than one task has issued a receive, the message goes to the highest priority receiver. If all waiting receivers have the same priority, the task scheduler selects the receiver.

Note that ?XMT may cause immediate task rescheduling if there is a waiting receiver with a higher priority than the ?XMT caller.

Notes

- See the descriptions of ?XMTW, ?REC, and ?RECNW in this chapter.

?XMTW — Transmits an intertask message and waits for it to be received.

?XMTW
error return
normal return

Input		Output	
AC0	Address of the mailbox that is to receive the intertask message	AC0	Unchanged
AC1	Message	AC1	Unchanged
AC2	To broadcast the message to all waiting receivers, –1	AC2	Unchanged

Error Codes in AC0

ERVWP	Invalid word pointer passed as a system call argument
ERXMT	Signal to address already in use (You tried to send a message to a mailbox that already contains a message.)
ERXMZ	Attempt to XMT illegal message

Why Use It?

You can use ?XMTW to synchronize the calling task with one or more receiving tasks. ?XMTW is more appropriate than ?XMT for this purpose, because it suspends the calling task until the receiver picks up the 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

Like ?XMT, ?XMTW allows the calling task to send a nonzero message to an empty mailbox and, ultimately, to the task or tasks that issue ?REC or ?RECNW system calls against that mailbox. (For 32–bit processes, the mailboxes are 32 bits wide; for 16–bit processes, the mailboxes are 16 bits wide.)

?XMTW differs from ?XMT in that it suspends the calling task until a receiving task issues either ?REC or ?RECNW against the mailbox. The operating system does not suspend the ?XMTW caller if there is an outstanding receive against the mailbox.

?XMTW takes the same input parameters as ?XMT. Note that ?XMTW also allows you to "broadcast" the message to all waiting receivers, by loading AC2 with –1 at input.

?XMTW may cause immediate task rescheduling if a waiting receiver has higher priority than the ?XMTW caller.

Notes

- See the descriptions of ?XMTW, ?REC, and ?RECNW in this chapter.

?XPSTAT Returns extended status information on a process.

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

Operating System Differences	
Accumulator	
Input and Output	Some
Error Codes	Some
Parameter Packet	Some

Input

AC0 One of the following:

- PID of the target process
- Byte pointer to the name of the target process
- –1 for status information about the calling process

AC1 One of the following:

- –1 if AC0 is a byte pointer
- Any other value if AC0 contains either –1, a PID, or a VPID

AC2 Address of the ?XPSTAT packet, unless you specify the address as an argument to ?XPSTAT

Output

AC0 Unchanged

AC1 Unchanged

AC2 Address of the ?XPSTAT packet

Error Codes in AC0

ERIRB Insufficient room in buffer (for username, ?PROC–related username, memordescriptors, 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
ER_FS_GROUP_BUFFER_TOO_SMALL
Size of buffer is too small to store the group access control list (AOS/VS II)

Why Use It?

Like ?PSTAT, ?RUNTM, and ?WHIST, ?XPSTAT 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 ?XPSTAT 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.

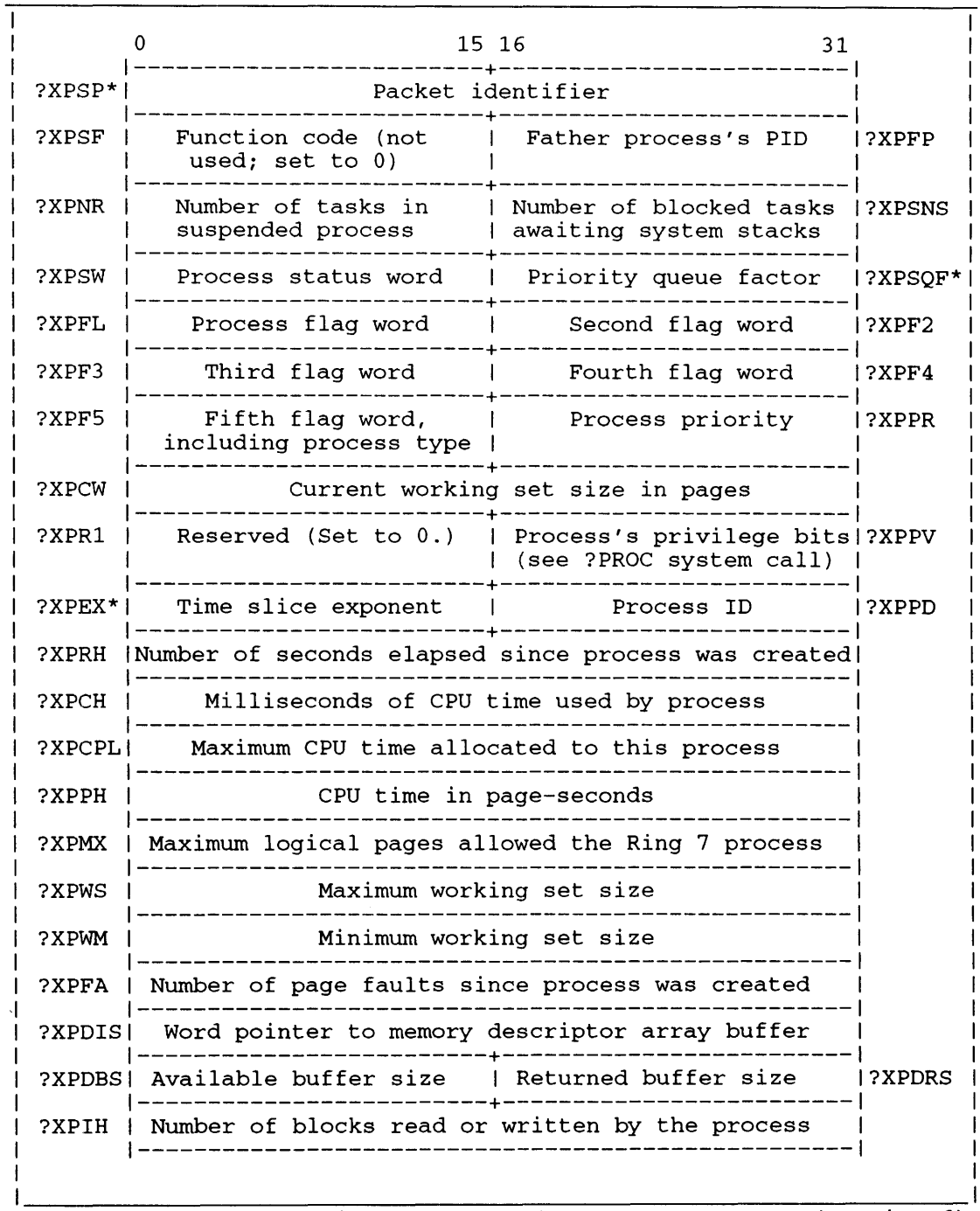
What It Does

?XPSTAT 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 2–278 shows the structure and Table 2–237 describes the contents of the ?XPSTAT packet.

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 2–279 shows the standard memory descriptor structure. Figure 2–280 shows the extended memory descriptor structure for AOS/VS and AOS/VS II.

?XPSTAT provides most of the information that ?PSTAT provides. Only the offsets from ?PSSN through ?PSEN (?PSAL words) are in ?PSTAT alone. These words return information about children PIDs 1–255. However, system call ?SONS provides you with the information in these offsets — and with information about son PIDs greater than 255.

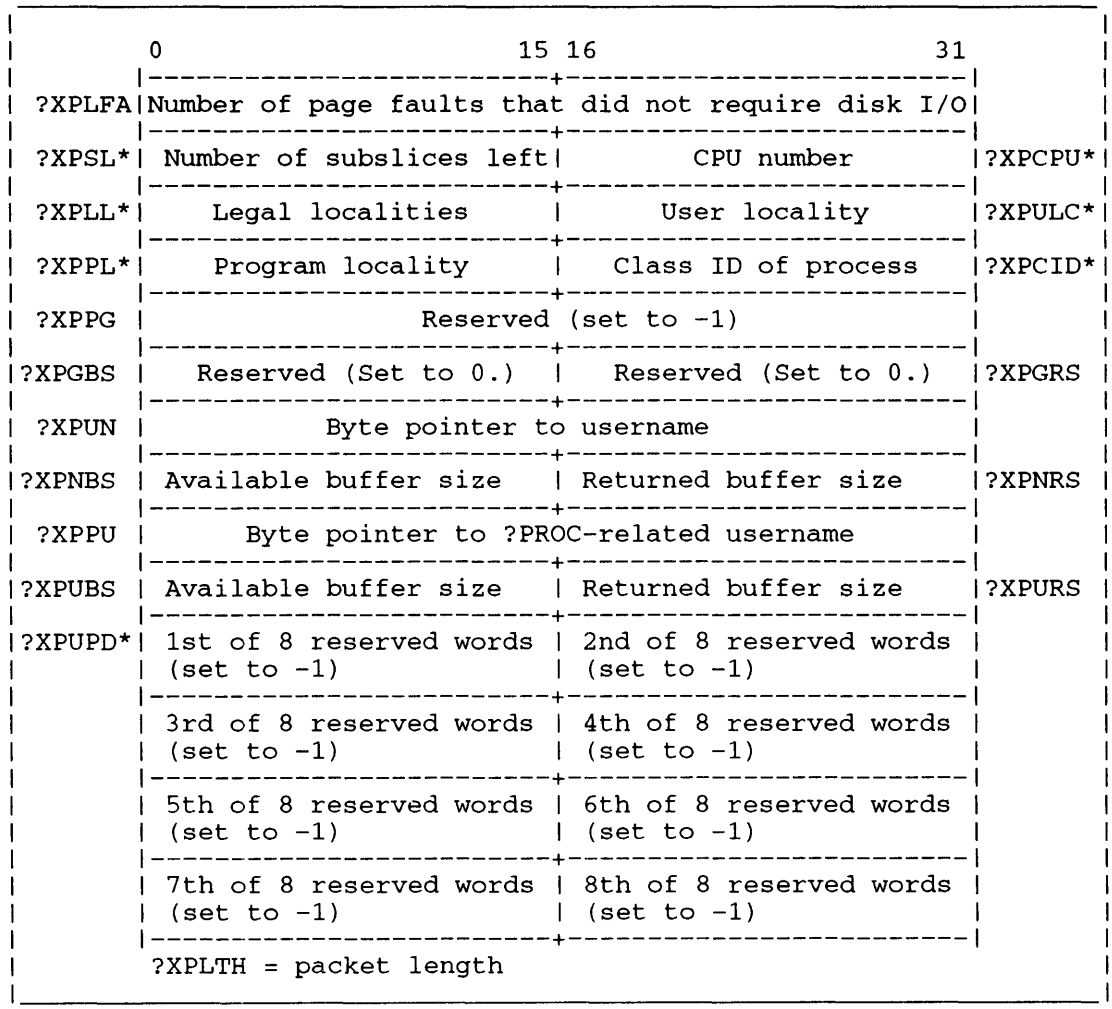
?XPSTAT also provides you with information in the ?PROC system call's extended packet, such as current user locality and legal localities. So, we recommend that you use ?XPSTAT and/or ?SONS in your new programs since (together) they have all the functionality of ?PSTAT — and more. ?XPSTAT also executes considerably faster than ?PSTAT.



* This offset differs in AOS/VS, AOS/VS II, and AOS/RT32. See Table 2-237.

(continued)

Figure 2-278. Structure of ?XPSTAT Packet



* This offset differs in AOS/VS AOS/VS II, and AOS/RT32. See Table 2-237. (concluded)

Figure 2-278. Structure of ?XPSTAT Packet

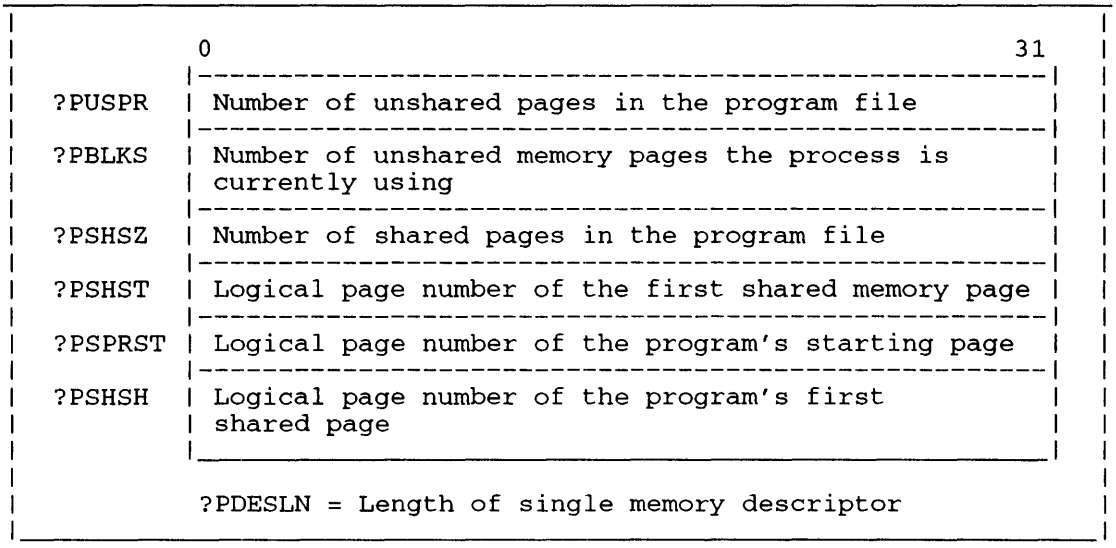


Figure 2–279. Structure of ?XPSTAT Standard Memory Descriptor

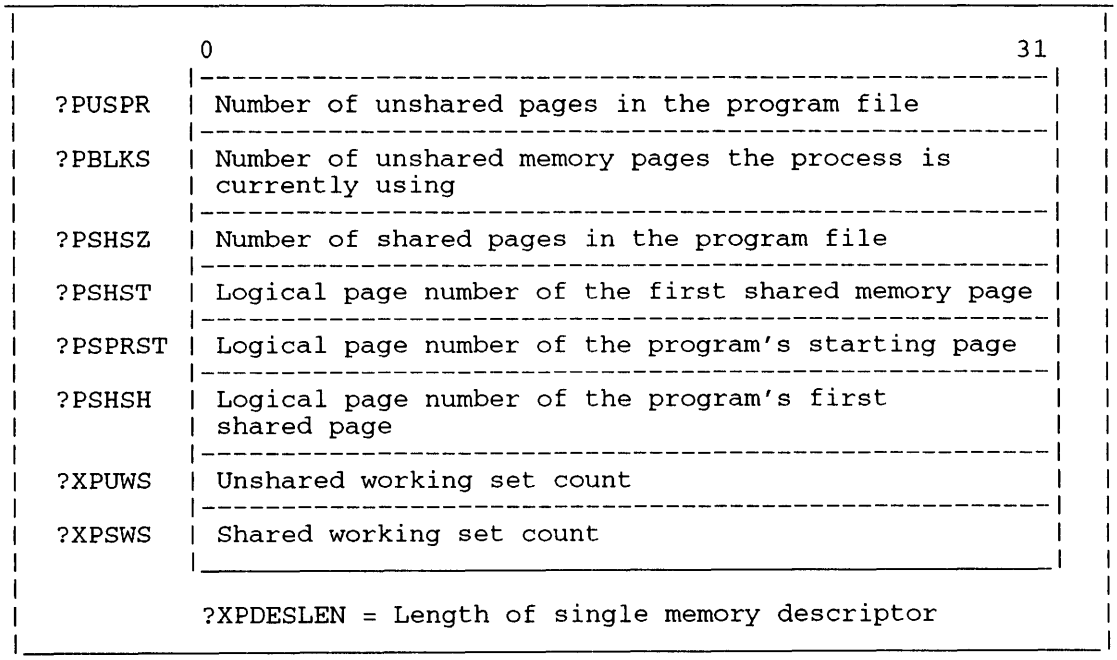


Figure 2–280. Structure of ?XPSTAT Extended Memory Descriptor (AOS/VS and AOS/VS II)

In Table 2–237, the column heads "VS" and "RT" represent AOS/VS and AOS/RT32 respectively. The entries in these columns have the following codes:

——	An operating system does not define a value for the given offset.
U	An operating system handles the offset differently (uniquely).
[no entry]	The operating systems handle the offset in the same way.

Table 2–237. Contents of ?XPSTAT Standard Parameter Packet

Offset	VS	RT	Description
?XPSP	U		Packet identifier: ?XPSID for the standard packet ?XPSID1 for extended packet with extended memory descriptors (AOS/VS and AOS/VS II) ?XPSID2 for extended packet with standard memory descriptors (AOS/VS II only)
?XPSF			Function code. Place 0 here.
?XPFP			PID or VPID 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.
?XPNR			Number of tasks in the process currently suspended on ?IREC system calls.
?XPSNS			Number of tasks that are blocked awaiting system stacks.
?XPSW			Process status word (Bit 6: process is blocked).
?XPSQF		---	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.
?XPFL			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
?XPF2			Second flag word.

(continued)

Table 2–237. Contents of ?XPSTAT Standard Parameter Packet

Offset	VS	RT	Description
?XPF3			Third flag word.
?XPF4			Fourth flag word.
?XPF5			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
?XPPR			Process priority.
?XPCW			Current number of shared and unshared pages in process's logical address space (the OS user-defined "working set" size).
?XPR1			Reserved. (Set to 0.)
?XPPV			Process's privilege bits. (See ?PROC in this system call dictionary for information on access privileges.)
?XPEX	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 ?XPSL in this table and the functional description below.)
?XPPD			PID or VPID of the process.
?XPRH			Number of seconds that have elapsed since the process was created.
?XPCH			CPU time (in milliseconds) used by the process.
?XPCPL			Maximum CPU time allocated to this process.
?XPPH			CPU time (in page-seconds).
?XPMX			Maximum number of logical pages allowed the process in Ring 7.
?XPWS			Maximum working set size in pages.
?XPWM			Minimum working set size in pages.

(continued)

Table 2–237. Contents of ?XPSTAT Standard Parameter Packet

Offset	VS	RT	Description
?XPFA			Number of page faults since the process was created (with ?PROC).
?XPDIS			Word pointer to a buffer that receives the memory descriptor array. See Figure 2-279 or 2-280. If you supply -1, the operating system returns nothing in the buffer.
?XPDBS			Size of the buffer, in words, that ?XPDIS identifies: Supply a 0 here if you specified a -1 in offset ?XPDIS. Supply at least 7*?PDESLN for packet revisions ?XPSID and ?XPSID2 (in offset ?XPSP). Supply at least 7*?XPDESLN for packet revision ?XPSID1 (in offset ?XPSP).
?XPDRS			Actual number of words returned in the buffer that ?XPDIS identifies. This is 7*?PDESLN or 7*?XPDESLN.
?XPIH			Number of blocks read or written by the process.
?XPLFA			Number of page faults that did not require disk input/output.
?XPSL	U	U	Number of remaining subslices in the process's time slice. This value is based on the current time slice exponent returned in ?XPEX. If the time slice exponent is 1, an AOS/VS process has 2 remaining subslices (that is, 2**1). AOS/RT32 does not break time slices into subslices, so it returns 0.
?XPCPU		---	CPU number. AOS/RT32 does not return this number.
?XPLL		---	Legal localities. AOS/RT32 does not return this number.
?XPULC		---	Current user locality. AOS/RT32 does not return this number.
?XPPL		---	Program locality. AOS/RT32 does not return this number.
?XPCID		---	Class ID of the process. AOS/RT32 does not return this number.
?XPPG			Reserved. (Set to -1.)
?XPGBS			Reserved. (Set to 0.)
?XPGRS			Reserved. (Set to 0.)

(continued)

Table 2–237. Contents of ?XPSTAT Standard Parameter Packet

Offset	VS	RT	Description
?XPUN			Byte pointer to the buffer that receives the username. If you supply -1, the operating system returns nothing in the buffer.
?XPNBS			Number of bytes in the buffer that ?XPUN points to. It must be at least ?MXUN, unless you supply -1 in offset ?XPUN.
?XPNRS			Actual number of bytes returned in the buffer (including the terminating null) that ?XPUN points to.
?XPPU			Byte pointer to the buffer that receives the ?PROC-related username. If you supply -1, the operating system returns nothing in the buffer.
?XPUBS			Number of bytes in the buffer that ?XPPU points to. It must be at least ?MXUN, unless you supply -1 in offset ?XPPU.
?XPURS			Actual number of bytes returned in the buffer (including the terminating null) that ?XPPU points to.
?XPUPD (8 words)	U	---	For AOS/RT32, reserved. (Set to 0.) For AOS/VS and AOS/VS II (extended packet), redefined. See Table 2-238 and Figure 2-281.

(concluded)

Table 2–238. Contents of ?XPSTAT Extension Packet for AOS/VS and AOS/VS II

Offset	Description
?XPSP to ?XPURS	See Table 2-237.
?XPGLT	Byte pointer to the group buffer, which contains the group access control list. If set to -1, the list is not returned.(AOS/VS II) For AOS/VS, reserved. (Set to 0.)
?XPGAB	Length of the buffer that ?XPGLT identifies. It must be at least ?GROUP_MIN_BUFFER_BYTE_LENGTH to ?GROUP_MIN_BUFFER_BYTE_LENGTH, unless you specify -1 in offset ?XPGLT. (AOS/VS II) For AOS/VS, reserved. (Set to 0.)
?XPGRB	Actual number of bytes returned in the group buffer, including the terminating double nulls. (AOS/VS II). For AOS/VS, reserved (Set to 0.)
?XPUQSH	Returns the number of shared pages in all rings unique to this process. (AOS/VS and AOS/VS II)

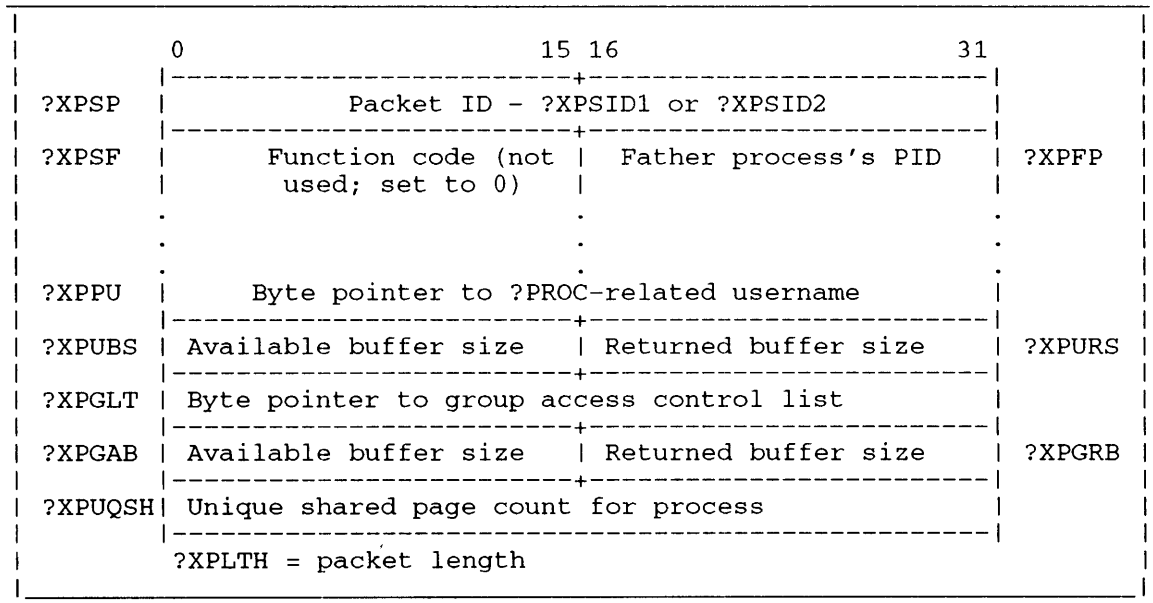


Figure 2–281. Structure of ?XPSTAT Extension Packet for AOS/VS and AOS/VS II

Notes

- See the descriptions of ?PSTAT, ?RUNTM, and ?WHIST in this chapter.
- See the description of ?PROC in this chapter, which initializes the group access control list for a process.
- See the description of ?GROUP in this chapter, which changes the group access control list for a process.
- See the description of ?XGTACP in this chapter, which returns the access control for a specific file, username, and group access control list.
- The parameters shown in Figure 2–279 are offsets from the memory descriptor base address. To get the base address for a particular ring, use the following formula:

  <address_returned_in_offset_?XPDIS> + (?PDESLN * (ring_number–1))

  where

  ?XPDIS — is the offset in the ?PSTAT packet that contains the address of 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:

  <address_returned_in_offset_?XPDIS> + (?PDESLN* 4)

End of Chapter

Appendix A
Sample Programs

This appendix 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.

Table A–1 lists the names of the programs in each program set along with the major system concepts and calls that the program set illustrates. Complete explanations of each program set, with program listings, appear after Table A–1.

The source program files, except for the FORTRAN 77 ones, are part of each release of AOS/VS. Their location is subdirectory :UTIL:SYSTEM_CALL_SAMPLES. We encourage you to create the program (.PR) files and obtain results equivalent to those in this appendix. We also encourage you to experiment with the programs by changing them and observing the effects of the changes. If your system has FORTRAN 77 software you can key in the relatively small FORTRAN 77 source program files.

Table A–1. Sample Program Sets and their Descriptions

Program Set Number and Program Names	Summary Description
1. HEAR.SR, SPEAK.SR, SON.SR (subroutine)	These source program units produce two executable programs that illustrate inter-process communication. The program units' respective system calls are ?CREATE, ?ILKUP, ?IREC, ?OPEN, ?RETURN, ?WDELAY, ?WRITE; ?CREATE, ?ILKUP, ?ISEND, ?RETURN ?TERM, ?WDELAY; and PROC, ?RETURN.
2. RUNTIME.SR	This program unit obtains its process's runtime statistics via system call ?RUNTM. It also issues system calls ?READ, ?RETURN, and ?WRITE.

(continued)

Table A–1. Sample Program Sets and their Descriptions

Program Set Number and Program Names	Summary Description
3. RINGLOAD.SR, INRING.SR, GATE.ARRAY.SR (subroutine)	These source program units produce two executable programs. RINGLOAD.PR loads INRING.PR into an inner ring and executes it -- illustrating instruction LCALL, how to define a gate array, and system call ?RINGLD. The program units' respective system calls are ?OPEN, ?RETURN, ?RINGLD, ?WRITE; ?DEBUG, ?OPEN, ?WRITE; and none -- GATE.ARRAY issues no system calls.
4. FILCREATE.SR	It opens the terminal and creates a disk file, showing system calls ?CREATE, ?DELETE, ?OPEN, ?READ, ?RETURN, and ?WRITE.
5. WRITE.SR	It opens the terminal and a disk file, gets text from the terminal, writes it to the disk file, and plays back the text on demand. The related system calls are ?OPEN, ?READ, ?WRITE, and -- to show file positioning -- ?SPOS. The program also issues system calls ?CLOSE and ?RETURN.
6. DLIST.SR	It lists all filenames in a directory to the line printer. It shows calls ?GTMES, ?RETURN, and ?SEND plus I/O via calls ?GNFN, ?GOPEN, ?OPEN, and ?WRITE.
7. NEWTASK.SR	This program shows a simple use of multi-tasking. It issues system calls ?IDKIL, ?MYTID, ?OPEN, ?RETURN, ?TASK, and ?WRITE.
8. BOOMER.SR	Here is a fast multitasked file copy program that uses two synchronized tasks to read and write data. It issues system calls ?GTMES, ?KILL, ?OPEN, ?READ, ?REC, ?RETURN, ?TASK, ?WRITE, and ?XMTW.
9. TIMEOUT.SR	This program delays itself for a specified period. It illustrates system calls ?GTMES, ?RETURN, and ?WDELAY.
10. DIRCREATE.F77, CHECK.F77 (subroutine)	This program is a FORTRAN 77 equivalent of the sample assembly language program in Chapter 1. DIRCREATE.PR issues ?CREATE (directory version) and ?DIR system calls. CHECK.F77 issues system call ?RETURN.
11. CREATE_ WINDOW.SR	This program uses calls ?PROC and ?WINDOW as it creates and deletes a window.
12. GRAPHICS_ SAMPLE.SR	This program creates a window and uses a pointer device to draw lines and bars. The system calls that it issues are ?GRAPHICS, ?OPEN, ?PTRDEVICE, ?READ, ?RETURN, and ?WINDOW.

(concluded)

Program Set 1 — HEAR.SR, SPEAK.SR, SON.SR

SON.SR is a subroutine that issues ?PROC to create and start a process. HEAR.SR/SON.SR and SPEAK.SR form two complete programs — HEAR.PR and SPEAK.PR — that illustrate interprocess communications. Program HEAR uses subroutine SON to execute program SPEAK. HEAR then issues an ?IREC call for a message from SPEAK, and SPEAK sends the message via an ?ISEND call. Each program uses ?ILKUP to discover the other's port number.

Figure A–1 contains HEAR.SR, Figure A–2 contains SPEAK.SR, and Figure A–3 contains SON.SR.

```
; This program module, HEAR.SR, calls module
;    SON to create and run program SPEAK.PR.
;    SPEAK then tries to send an IPC message to HEAR.
;    HEAR's destination IPC port is PORTR; its
;    origin port is PORTS.  After program SPEAK
;    sends the message, it terminates; then
;    program HEAR passes SPEAK's  message to the
;    CLI on termination.  The Link lines are
;
;       X  LINK HEAR SON    and
;       X  LINK SPEAK
;
; To use routine SON, you need "Create without
;    Block", "Use IPC", and "SONs" privileges.
        .TITLE  HEAR
        .EXTN   SON         ; External address.
        .ENT    HEAR        ; Begin here.
        .NREL               ; Default to partition 4.
        ; Open terminal for input and output.
HEAR:   ?OPEN   CONSOLE     ; For Input and Output.
        WBR     E.RROR      ; Error, process it.
        ?WRITE  CONSOLE     ; Write message (byte pointer is
                            ;    already in I/O packet.)
        WBR     E.RROR      ; Error, process it.
        ; Start the SON process to run SPEAK.PR.
        XLEFB   0, SPEAK*2 ; Get byte pointer to filename.
        XJSR    SON         ; SON creates process.
        ; SPEAK is running.  Create IPC entry for receive.
        XLEFB   0,PORTR*2   ; Byte pointer to port name.
        ?CREATE  IPCEN      ; Create IPC entry PORTR.
        WBR     E.RROR      ; Error, process it.
        XLEFB   0, MES1*2   ; Byte pointer to text message.
        XWSTA   0, CONSOLE+?IBAD ; Put in I/O packet.
        ?WRITE  CONSOLE     ; Write message.
        WBR     E.RROR      ; Error, process it.
```

Figure A–1. Listing of Program HEAR.SR (continued)

```
        ; See if SPEAK's entry, PORTS, has been created.
GETOP:  XLEFB   0, PORTS*2 ; Byte pointer to port name.
        ?ILKUP             ; Get port number in AC1.
        WBR     TEST       ; Error -- check for
                           ;    ERDNE (doesn't exist).
        XLEFB   0, MES2*2  ; Entry exists, get byte pointer.
        XWSTA   0, CONSOLE+?IBAD  ; Put in I/O packet.
        ?WRITE  CONSOLE    ; Write "Success" message.
        WBR     E.RROR     ; Error, process it.

        XWSTA   1, RHDR+?IOPH  ; Put origin port number in
                           ;        receive header (Note
                           ;        Wide Store instruction.)
        NLDAI   1, 0       ; Generate 1.
        XNSTA   0, RHDR+?IDPN  ; Put destination port 1 in
                           ;        receive header (Note
                           ;        Narrow Store instruction.)
        ?IREC   RHDR       ; Receive SPEAK message.
        WBR     E.RROR     ; Error, process it.
        XLEFB   1, MSBUF*2 ; Message received; get byte pointer
                           ;    to message buffer.
        NLDAI   ?RFCF+100., 2  ; Put flag + 100. words for
                           ;        message in AC2.
        ?RETURN            ; Return with message to CLI.
        WBR     E.RROR     ; Return error.

        ; ?ILKUP error. Check for "does not exist" error
        ;     and delay if this was the error.
TEST:   NLDAI   ERFDE, 1   ; Put ERFDE number in AC1.
        WSEQ    0, 1       ; Skip if ERFDE.
        WBR     E.RROR     ; Handle other error.
        XLEFB   0, MES3*2  ; ERFDE. Get message byte pointer.
        XWSTA   0, CONSOLE+?IBAD ; Put in I/O packet.
        ?WRITE  CONSOLE    ; Write message.
        WBR     E.RROR     ; Error, process it.

        NLDAI   5000., 0   ; 5000 milliseconds = 5 seconds.
        ?WDELAY            ; Delay for 5 seconds.
        WBR     E.RROR     ; Error, process it.
        WBR     GETOP      ; Try ?ILKUP again.

        ; Error instructions, pointer, file/port names.
E.RROR: XJMP    ERROR      ; To error handler.
;.SON:  SON                ; To subroutine SON.

SPEAK:  .TXT  "SPEAK.PR"   ; SPEAK program filename.
PORTR:  .TXT  "PORTR"      ; Receive port name.
PORTS:  .TXT  "PORTS"      ; Send port name.
```

Figure A–1. Listing of Program HEAR.SR (continued)

```
        ; Set up open and I/O packet for terminal.
        .ENABLE WORD        ; Most entries = 16 bits.
                            ; Use WORD for rest of program.
CONSOLE: .BLK   ?IBLT       ; Packet length.
        .LOC    CONSOLE+?ISTI
        ?ICRF+?RTDS+?OFIO   ; Data-sensitive records,
                            ;   input and output.
        .LOC    CONSOLE+?IMRS
         -1     ; Memory block size, default to 2048.
        .LOC    CONSOLE+?IBAD
        .DWORD  MES*2       ; 2-word byte pointer to text/buffer.
        .LOC    CONSOLE+?IRCL
        120.    ; Maximum record length of 120 characters.
        .LOC    CONSOLE+?IFNP
        .DWORD  CON*2       ; 2-word byte pointer to filename.
        .LOC    CONSOLE+?IDEL
        .DWORD  -1          ; Data-sensitive delimiter table address;
                            ;    default to NL, CR, NUL.
        .LOC    CONSOLE+?IBLT ; Default others to 0.
        ; End of console I/O packet.
; Filename, buffer, messages. .NOLOC 1 follows ...
        .NOLOC  1           ;  ... to save listing space.
CON:    .TXT "@CONSOLE"     ; Use generic name.
MES:    .TXT "From HEAR -- have opened terminal; ready to call SON.<12>"
MES1:   .TXT "From HEAR -- back from SON, SPEAK is running. Created
 IPC entry.<12>"
MES2:   .TXT "From HEAR -- have ?ILKUPed the IPC port entry.<12>"
MES3:   .TXT "From HEAR -- Error on ?ILKUP. Will delay and
 try again.<12>"
        .NOLOC  0           ; Resume listing all.
```

Figure A–1. Listing of Program HEAR.SR (continued)

```
; Header for IPC entry.
IPCEN:  .LOC    IPCEN+?CFTYP
        ?FIPC               ; FIPC type file.
        .LOC    IPCEN+?CPOR
        1                   ; Port number is 1.
        .LOC    IPCEN+?CTIM
        .DWORD  -1          ; Default to current time.
        .LOC    IPCEN+?CACP
        .DWORD  -1          ; Default to current ACL.
                            ; Receive header RHDR -- for the ?IREC.
RHDR:   .LOC    RHDR+?ISFL
        0                   ; No system flags.
        .LOC    RHDR+?IUFL
        0                   ; No user flags.
        .LOC    RHDR+?IOPH
        .DWORD  0           ; For origin port number.
        .LOC    RHDR+?IDPN
        0                   ; For destination port number.
        .LOC    RHDR+?ILTH
        100.                ; Message buffer, 100. words.
        .LOC    RHDR+?IPTR
        .DWORD  MSBUF       ; Message buffer address.
        .LOC    RHDR+?PLTH  ; End of header.
MSBUF:  .BLK    101.        ; Message buffer.
        ;  Error handler.
ERROR:  NLDAI   ?RFEC+?RFCF+?RFER, 2  ; Error flags.
        ?RETURN             ; To CLI.
        WBR     ERROR       ; Return error.
        .END    HEAR
```

Figure A–1. Listing of Program HEAR.SR (concluded)

```
        .TITLE  SPEAK
        .ENT    SPEAK
        .NREL
; The sole purpose of this program is to send
;    an IPC message to a brother process, then to
;    terminate itself. SPEAK's origin port is
;    PORTS and its destination port is PORTR.
        ; Create an IPC entry port named PORTS.
SPEAK:  XLEFB   0, PORTS*2 ; Byte pointer to port name.
        ?CREATE IPCEN      ; Create IPC port.
        WBR     E.RROR     ; Error, process it.
        ; See if receive entry, PORTR, has been created.
GETNM:  XLEFB   0, PORTR*2 ; Byte pointer to port name.
        ?ILKUP             ; Put port number in AC1.
        WBR     TEST       ; Error -- check for
                           ; ERDNE (doesn't exist).
        XWSTA   1, SHDR+?IDPH  ; Entry exists; put
                               ;    number in send header.
        NLDAI   1, 0           ; Generate 1.
        XNSTA   0, SHDR+?IOPN  ; Put destination port 1
                               ;    in send header. Note
                               ;    narrow store instruction.
        ?ISEND  SHDR           ; Send SPEAK message.
        WBR     E.RROR         ; Error, process it.
        ; Message sent. Delay to allow brother to receive
        ;    message before terminating yourself.
        NLDAI   10000.,  0     ; 10000 ms = 10 seconds.
        ?WDELAY                ; Delay for 10 seconds.
        WBR     E.RROR         ; Error, process it.
        NLDAI   -1, 0          ; Get -1 to terminate self.
        WSUB    2,2            ; No IPC message to father.
        ?TERM                  ; The end.
        WBR     E.RROR         ; Termination error, handle it.
        ; ?ILKUP error. Check for "does not exist" error and delay
        ;    if this was the error.
TEST:   NLDAI   ERFDE,  1      ; Put ERFDE number in AC1.
        WSEQ    0, 1           ; Skip if ERFDE.
        WBR     E.RROR         ; Handle other error.
        NLDAI   5000.,  0      ; 5000 milliseconds =  5 seconds.
        ?WDELAY                ; Delay for 5 seconds.
        WBR     E.RROR         ; Error, process it.
        WBR     GETNM          ; Try ?ILKUP again.
        ; Error instruction, pointer, file/port names.
E.RROR: XJMP    ERROR          ; To error handler.
PORTS:  .TXT    "PORTS"        ; Send port name.
PORTR:  .TXT    "PORTR"        ; Receive port name.
```

Figure A–2. Listing of Program SPEAK.SR (continued)

```
        ; Header for IPC entry.
        .ENABLE WORD            ; Most entries = 16 bits.
                                ;    Use WORD for rest of prog.
IPCEN:  .LOC    IPCEN+?CFTYP
        ?FIPC                   ; FIPC type file.
        .LOC    IPCEN+?CPOR
        1                       ; Port number is 1.
        .LOC    IPCEN+?CTIM
        .DWORD  -1              ; Default to current time.
        .LOC    IPCEN+?CACP
        .DWORD  -1              ; Default to current ACL.
        ; Send header SHDR -- for the ?ISEND.
SHDR:   .LOC    SHDR+?ISFL
        0                       ; No system flags.
        .LOC    SHDR+?IUFL
        0                       ; No user flags.
        .LOC    SHDR+?IDPH
        .DWORD  0               ; For destination port number.
        .LOC    SHDR+?IOPN
        0                       ; For origin port number.
        .LOC    SHDR+?ILTH
        100.                    ; Message buffer, 100. words.
        .LOC    SHDR+?IPTR
        .DWORD  MSBUF           ; Message buffer address.
        .LOC    SHDR+?PLTH      ; End of header.
        ; Message to send. A .NOLOC 1 follows ...
        .NOLOC  1               ; ... to save listing space.
       MSBUF:  .TXT  "Hello, this is your son SPEAKing. As you <12>
read these words, I am terminating and so are you.<12>"
        .NOLOC  0               ; Resume listing all.
        ;  Error handler.
ERROR:  NLDAI   ?RFEC+?RFCF+?RFER,  2  ; Error flags.
        ?RETURN                 ; To CLI.
        WBR     ERROR           ; Return error.
        .END    SPEAK
```

Figure A–2. Listing of Program SPEAK.SR (concluded)

```
        .TITLE  SON
        .ENT    SON
        .NREL
; This subroutine, SON.SR, creates a swappable
;   son process. The process runs program SPEAK.PR,
;   but you can change this by replacing SPEAK.PR
;   with any executable-program-name.PR.  All
;   defaultable ?PROC characteristics are
;   defaulted: No IPC header, processname is its
;   PID, same maximum size as the calling process,
;   same generic files and username and privileges
;   as the father, no sons. To use SON, you must
;   have "Create without Block" privilege in your
;   user profile.
;  Get program name to ?PROC.
SON:    WSSVR   0               ; Save return from XJSR.
        XLEFB   0, PRGNM*2      ; Byte pointer to program name.
        XWSTA   0,PKT+?PSNM     ; Put in ?PROC packet.
        ?PROC   PKT             ; Create process.
        WBR     ERROR           ; Error, handle it.
        WRTN                    ; To caller.
PRGNM:  .TXT    "SPEAK.PR"
ERROR:  NLDAI   ?RFEC+?RFCF+?RFER,2  ; Error flags.
        ?RETURN                 ; To CLI.
        WBR     ERROR           ; Return error.
        ; ?PROC packet.
        .ENABLE WORD            ; Most entries = 16 bits.
       PKT:
        .BLK    ?PLTH           ; Packet length.
        .LOC    PKT+?PFLG
        0                       ;  Default creation specifications.
        .LOC    PKT+?PPRI
        -1                      ; Priority (default = caller's).
        .LOC    PKT+?PSNM
        .DWORD  PRGNM*2         ; 2-word byte pointer to process name.
        .LOC    PKT+?PIPC
        .DWORD  -1              ; 2-word address of IPC message header;
                                ;    there is no header.
        .LOC    PKT+?PNM
        .DWORD  -1              ; 2-word byte pointer to process name;
                                ;    default = PID.
        .LOC    PKT+?PMEM
        .DWORD  -1              ; Maximum memory pages;
                                ;    default = caller's.
        .LOC    PKT+?PDIR
        .DWORD  -1              ; 2-word byte pointer to directory;
                                ;    default = working directory.
        .LOC    PKT+?PCON
        .DWORD  0               ; 2-word byte pointer to @CON name.  None.
```

Figure A–3. Listing of Program SON.SR (continued)

```
.LOC    PKT+?PCAL
-1                      ; Default = 2 concurrent system calls.
.LOC    PKT+?PWSS
-1                      ; Maximum working set; default=caller's.
.LOC    PKT+?PUNM
.DWORD  -1              ; Username; default = caller's.
.LOC    PKT+?PPRV
?PVIP                   ; Privileges; allow ?ISEND call.
.LOC    PKT+?PPCR
0                       ; Number of processes son can create.
.LOC    PKT+?PWMI
-1                      ; Minimum working set; default=caller's.
.LOC    PKT+?PIFP
.DWORD  0               ; 2-word byte pointer to @INPUT name;
                        ;    none.
.LOC    PKT+?POFP
.DWORD  0               ; 2-word byte pointer to @OUTPUT name;
                        ;    none.
.LOC    PKT+?PLFP
.DWORD  0               ; 2-word byte pointer to @LISTFILE name;
                        ;    none.
.LOC    PKT+?PDFP
.DWORD  0               ; 2-word byte pointer to @DATAFILE name;
                        ;    none.
.LOC    PKT+?SMCH
.DWORD  -1              ; Maximum CPU time; default to father's.
.LOC    PKT+?PLTH
; End of ?PROC packet.
.END
```

Figure A–3. Listing of Program SON.SR (concluded)

The assembly commands are

```
) XEQ MASM HEAR ↵
) XEQ MASM SON ↵
) XEQ MASM SPEAK ↵
```

The Link commands are

```
) XEQ LINK HEAR SON ↵
) XEQ LINK SPEAK ↵
```

The command to execute both programs is

```
) XEQ HEAR ↵
```

You need USE IPC and CREATE WITHOUT BLOCK privileges in your user profile to execute the programs.

The console output from the execution command is next.

From HEAR — have opened terminal; ready to call SON.
From HEAR — back from SON, SPEAK is running. Created IPC entry.
From HEAR — Error on ?ILKUP. Will delay and try again.
From HEAR — have ?ILKUPed the IPC port entry.

Hello, this is your son SPEAKing. As you
read these words, I am terminating and so are you.

Program Set 2 — RUNTIME.SR

Program RUNTIME uses system call ?RUNTM to get two of its own runtime statistics. They are the process's CPU time in milliseconds (offset ?GRCH) and CPU usage in page–seconds (offset ?GRPH). Then, RUNTIME converts them to a string of ASCII decimal digits, displays them, and loops. The program continues this ?RUNTM–display loop until you type the three–character terminating sequence ST<New Line>.

Figure A–4 contains RUNTIME.SR.

```
; This routine gets and displays two runtime statistics
;    on a process. It opens the terminal, issues ?RUNTM,
;    converts and displays the statistics. It gets its
;    own statistics, but you can have it get any process's
;    statistics by passing the process's filename.PR or
;    PID. To use it as a subroutine, start with proper Save,
;    and end with proper Return.
        .TITLE  RUNTIME
        .ENT    RUNTIME, CONVERT
        .NREL
        ; Open terminal for I/O.
RUNTIME:?OPEN   CONSOLE ; Open for input and output,
                        ;    with start message.
        WBR     ERROR   ; Error, handle it.
        ?WRITE  CONSOLE ; Write initial message.
        WBR     ERROR   ; Error, handle it.
        ; Call ?RUNTM to get statistics. (For other process'
        ; statistics, put name or PID in AC0 and -1 or 0 in AC1.)
LOOP:   NLDAI   -1, 0     ; Check self.
        ?RUNTM  RPKT      ; Get statistics in packet.
        WBR     ERROR     ; Error, process it.
        XWLDA   1, MSECS  ; Get milliseconds from RPKT.
        XLEFB   2, MSECMSG*2     ; Millisecond message byte address.
        XJSR    CONVERT   ; Convert milliseconds to ASCII
                          ;    decimal digits, put them in message.
        XLEFB   0,MSECMSG*2      ; Message done, get byte pointer.
        XWSTA   0,CONSOLE+?IBAD ; Put in I/O packet.
        ?WRITE  CONSOLE   ; Write milliseconds message.
        WBR     ERROR     ; Error, process it.
        XWLDA   1,PSECS   ; Get page-seconds from RPKT.
        XLEFB   2, PSECMSG*2     ; Page-seconds message byte address.
        XJSR    CONVERT   ; Convert value to ASCII
                          ;    decimal digits, put them in message.
        XLEFB   0,PSECMSG*2      ; Message done, get byte pointer.
        XWSTA   0,CONSOLE+?IBAD ; Put in I/O packet.
        ?WRITE  CONSOLE   ; Write page-seconds message.
        WBR     ERROR     ; Error, process it.
```

Figure A–4. Listing of Program RUNTIME.SR (continued)

```
        ; See if user wants to stop.
        XLEFB   0, BUF*2 ; Byte pointer to I/O buffer.
        XWSTA   0,CONSOLE+?IBAD ; Put in I/O packet.
        ?READ   CONSOLE  ; Read for terminator.
        WBR     ERROR    ; Process error.
        NLDAI   'ST', 0  ; Put 'ST' in AC0.
        XNLDA   1, BUF   ; Put first word of buffer in AC1.
        WSNE    0,1      ; Skip if first word not 'ST'.
        WBR     BYE      ; Good bye.
        WBR     LOOP     ; Do it all again.

        ;  Error handler and return.

ERROR:  NLDAI   ?RFEC+?RFCF+?RFER,  2  ; Error flags.
BYE:    WSUB    2,2             ; Good return flgs.
        ?RETURN                 ; To father.
        WBR     ERROR           ; Return error.

        ; Open and I/O packet for console.

        .ENABLE WORD
        ; Use WORD for rest of program.

CONSOLE: .BLK   ?IBLT           ; Packet length.
        .LOC    CONSOLE+?ISTI
        ?ICRF+?RTDS+?OFIO       ; Data-sensitive records, Input/Output.
        .LOC    CONSOLE+?IMRS
         -1                     ; Memory block size, default to 2048.
        .LOC    CONSOLE+?IBAD
        .DWORD  ITEXT*2         ; 2-word byte pointer to text/buffer.
        .LOC    CONSOLE+?IRCL
        120.                    ; Maximum record length of 120 chars.
        .LOC    CONSOLE+?IFNP
        .DWORD  CON*2           ; 2-word byte pointer to filename.
        .LOC    CONSOLE+?IDEL
        .DWORD  -1              ; Data-sensitive delimiter table
                                ;    address; default to NUL, NL, FF, CR.
        .LOC    CONSOLE+?IBLT   ; Default others to 0.

        ; End of console I/O packet.

        ; Filename, start message, buffer.  A .NOLOC 1 follows ...
        .NOLOC  1               ; ... to save listing space.
CON:    .TXT "@CONSOLE"         ; Use generic name.

ITEXT:  .TXT "I give runtime statistics on a process.
 Type ST[New Line] to return to father.<212><12>"

BUF:    .BLK (BUF-CON)*2        ; Use number of bytes in message.
        .NOLOC  0               ; Resume listing.

        ; Messages to include converted statistics. .NOLOC here ...
        .NOLOC  1               ; ... to save listing space.
MSECMSG: .TXT "        Milliseconds elapsed.<12>"

PSECMSG: .TXT "        Page-seconds used. Type
 ST[New Line] to stop, other char to loop.<212><12>"
        .NOLOC  0
```

Figure A–4. Listing of Program RUNTIME.SR (continued)

```
        ; ?RUNTM Packet.
RPKT:   .BLK    ?GRLTH          ; Length of packet.
SECS:   .LOC    RPKT+?GRRH
        .DWORD  0               ; System returns elapsed time in seconds.
MSECS:  .LOC    RPKT+?GRCH
        .DWORD  0               ; System returns CPU time in milliseconds.
IO:     .LOC    RPKT+?GRIH
        .DWORD  0               ; System returns I/O usage.
PSECS:  .LOC    RPKT+?GRPH
        .DWORD  0               ; System returns CPU time in page-seconds.
        .LOC    RPKT+?GRLTH     ; End of packet.
; Convert routine converts binary value to decimal and puts it in text
;    string. Enter with AC1 containing value, AC2 containing byte
;    address of text message.
CONVERT: WSSVS  0               ; Save return.
        WMOV    2,3             ; Use AC3 for byte pointer shifting.
        WADI    3,3             ; Add integer 3 to byte address.
        NLDAI   10., 2          ; Put 10. in AC2.
DLOOP:  WSUB    0,0             ; Zero AC0 (high-order part
                                ;    of dividend). AC1 still has
                                ;    low order part of dividend.
        WDIVS                   ; Divide by 10., put quotient
                                ;    in AC1, remainder in AC0.
        IORI    60,0            ; OR in 60 for ASCII number.
        WSTB    3, 0            ; Store AC0-byte (bits 24-31)
                                ;    in byte address of AC3.
        WSBI    1, 3            ; Decrement the byte address.
        MOV     1,1,SNR         ; Did quotient go to 0?
        WRTN                    ; Yes, done, return to caller.
        WBR     DLOOP           ; No, do another digit.
        .END    RUNTIME
```

Figure A–4. Listing of Program RUNTIME.SR (concluded)

The assembly command is

```
) XEQ MASM RUNTIME ↵
```

The Link command is

```
) XEQ LINK RUNTIME ↵
```

The command to execute the program is

```
) XEQ RUNTIME ↵
```

A typical line of console output from the execution command is next.

59 Milliseconds elapsed.
3 Page-seconds used. Type ST[NEW LINE] to stop, other char to loop.

Program Set 3 — RINGLOAD.SR, INRING.SR, and GATE.ARRAY.SR

These three source files form two programs that illustrate call ?RINGLD. The first source, RINGLOAD.SR, forms a complete program that executes in ring 7. The second module, INRING, must be linked with the third module, subroutine GATE.ARRAY. Together they form program file INRING.PR. RINGLOAD.PR loads INRING.PR into ring 5, where it executes and passes control to the debugger. After you type ESC–R to continue the program, control returns to RINGLOAD.PR in ring 7.

These programs show use of the pseudo–operation ?EXTG, the instruction LCALL, and the system call ?RINGLD.

Figure A–5 contains RINGLOAD.SR, Figure A–6 contains INRING.SR, and Figure A–7 contains GATE.ARRAY.SR.

```
; This program, RINGLOAD, loads program INRING
;    into an inner ring, then calls INRING via
;    LCALL. The inner ring can be 4, 5, or 6,
;    depending on the Link switch used on INRING.
;    RINGLOAD assumes that INRING.PR, linked from
;    INRING and GATE.ARRAY exists.

        .TITLE  RINGLOAD
        .ENT    RINGLOAD
        .NREL           ; Default unshared code.

; Open terminal, ?RINGLD program.

RINGLOAD: ?OPEN CONSOLE ; Open terminal for I/O.
        WBR     ERROR   ; Error, process it.
        ?WRITE  CONSOLE ; Write "About to.." message.
        WBR     ERROR   ; Error, process it.

        XLEFB   0, PNAME*2 ; Byte pointer to INRING name.
        ?RINGLD         ; Load INRING.
        WBR     ERROR   ; Error, process it.

; Do LCALL.  Real format is
;
;       LCALL displ index arg-count.
;
; Like a system call, LCALL has a single-word error return
;    location and double-word good return location:
;
;       LCALL  displ  index  arg-count
;       error return     (single-word)
;       normal return    (double-word)

        LCALL   INRING,0,0      ; Call it, index, argument count zero.
        WBR     INERROR         ; Report INRING error.

        ; Back from INRING. Depart with message for CLI.

        XLEFB   0, MES2*2       ;  Byte pointer to farewell message.
        XWSTA   0, CONSOLE+?IBAD ; Put in I/O packet.
        ?WRITE  CONSOLE         ; Write message.
        WBR     ERROR           ; Error, process it.
        WSUB    2, 2            ; Set for good return.
        WBR     BYE             ; Done, give message and depart.
```

Figure A–5. Listing of Program RINGLOAD.SR (continued)

```
        ; Inner program and current program error handlers.
INERROR: LLEFB  0, INMES*2       ; Get "Inner prog" error message byte ptr.
        LWSTA   0, CONSOLE+?IBAD ; Put in I/O packet.
        ?WRITE  CONSOLE          ; Write to terminal.
        WBR     ERROR            ; Write error.
ERROR:  WLDAI   ?RFEC+?RFCF+?RFER,  2  ; Error flags.
BYE:    ?RETURN                  ; To CLI.
        WBR     ERROR            ; Return error.
        ; Definition of inner program ring bracket and gate.
        ;    The system uses this, not program name (INRING)
        ;    to access gate and inner program.
INRING = 5S3+0                   ; Ring 5 + 1st gate.
        ; Messages. A .NOLOC 1 follows.
        .NOLOC  1
PNAME:  .TXT "INRING.PR"
MES1:   .TXT  "I'm RINGLOAD, about to ?RINGLD
 program INRING.<12>"
MES2:   .TXT "<212>I'm RINGLOAD, back from
 INRING, terminating.<12>"
INMES:  .TXT  "<212>Error in inner ring program.<12>"
        .NOLOC  0   ; Resume listing all.
        ; Open and I/O packet -- needed for I/O.
        .ENABLE WORD ; Most entries =16 bits.
CONSOLE: .BLK   ?IBLT            ; Packet length.
        .LOC    CONSOLE+?ISTI
        ?ICRF+?RTDS+?OFIO        ; Data-sensitive records, Input/Output.
        .LOC    CONSOLE+?IMRS
         -1                      ; Memory block size, default to 2048.
        .LOC    CONSOLE+?IBAD
        .DWORD  MES1*2           ; 2-word byte pointer to text/buffer.
        .LOC    CONSOLE+?IRCL
        120.                     ; Maximum record length of 120 chars.
        .LOC    CONSOLE+?IFNP
        .DWORD  CON*2            ; 2-word byte pointer to filename.
        .LOC    CONSOLE+?IDEL
        .DWORD  -1               ; Data sensitive delimiter table add; none.
        .LOC    CONSOLE+?IBLT    ; Default others to 0.
        ; End of console I/O packet.
CON:    .TXT  "@CONSOLE"
        .END    RINGLOAD
```

Figure A–5. Listing of Program RINGLOAD.SR (concluded)

```
; This program, INRING, is loaded by program
;    RINGLOAD, then called via LCALL. This program
;    saves the return address, opens the terminal,
;    writes  messages, invokes the debugger (so
;    that you can look around in inner ring). Then,
;    after you type ESC-R, it returns to RINGLOAD in
;    ring 7. All code is shared except for I/O packet.
;
; This program must be linked with module GATE.ARRAY.
;    It can be executed in ring 4, 5, or 6 depending on
;    three things: the LCALL name definition in the
;    calling program, the gate defined in GATE.ARRAY,
;    and the Link switch used on this program. Here, the
;    caller RINGLOAD defines the name as gate 5 (5S3);
;    GATE.ARRAY defines gate 5; and the Link command
;    used is    X LINK/RING=5 INRING GATE.ARRAY.

        .TITLE  INRING
        .ENT    INRING
        .NREL   1           ; Shared code (SC). Can be up
                            ;    to 1,024,000 bytes.

        ; Save return, open terminal, write message, go into
        ;    the debugger.

INRING: WSAVR   0           ; Save frame (4 ACs, PC in AC3)
        ?OPEN   CONSOLE     ; Open console for I/O.
        WBR     ERTN        ; Error, process it.
        ?WRITE  CONSOLE     ; Write message from ring 5.
        WBR     ERTN        ; Error, process it.
        ?DEBUG              ; Go into the debugger.
        WBR     ERTN        ; Error, process it.
        LLEFB   0, MES2*2 ; Get byte pointer to return message.
        LWSTA   0, CONSOLE+?IBAD ; Put in I/O packet.
        ?WRITE  CONSOLE     ; Write another message.
        WBR     ERTN        ; Error, process it.

        ; Done. Ready for good return to caller.

        LDAFP   3           ; Frame pointer in AC3.
        XWISZ   0, 3        ; Increment return address for good
                            ;    return to LCALLer.
        WRTN                ; Return to caller.

        ; INRING error handler. Returns error to the outer ring
        ;    caller, not to the CLI.

ERTN:   LDAFP   3           ; Get the frame pointer.
        LWSTA   0, ?OAC0, 3 ; Put error code (AC0)
                            ;       in saved frame's AC0.
                            ; In frame, ?OAC0 is offset of old AC0.
        WRTN                ; Return to LCALLer's error return

        ; Text messages.  .NOLOC 1 follows.
        .NOLOC  1
MES1:   .TXT  "I'm INRING, in inner ring. About
 to ?DEBUG. Type ESC R to proceed.<212><12>"

MES2:   .TXT  "<212><212>From INRING -- about
 to WRTN.<12>"
        .NOLOC  0           ; Resume listing all.

        .NREL               ; Must use unshared for packet because
                            ;    program and system write into it.
```

Figure A–6. Listing of Program INRING.SR (continued)

```
        ; Open and I/O packet for @CONSOLE.
         .ENABLE WORD      ; Most entries = 16 bits.
CONSOLE: .BLK    ?IBLT     ; Packet length.
        .LOC     CONSOLE+?ISTI
        ?ICRF+?RTDS+?OFIO ; Data-sensitive records, Input/Output.
        .LOC     CONSOLE+?IMRS
         -1               ; Memory block size, default to 2048.
        .LOC     CONSOLE+?IBAD
        .DWORD   MES1*2    ; 2-word byte pointer to text/buffer.
        .LOC     CONSOLE+?IRCL
        120.               ; Maximum record length of 120 characters.
        .LOC     CONSOLE+?IFNP
        .DWORD   CON*2     ; 2-word byte pointer to filename.
        .LOC     CONSOLE+?IDEL
        .DWORD   -1        ; Data-sensitive delimiter table address; default.
        .LOC     CONSOLE+?IBLT ; Default others to 0.
        ; End of console I/O packet.
        ; Filename. A .NOLOC 1 follows.
        .NOLOC   1
CON:    .TXT   "@CONSOLE"
        .NOLOC   0         ; Resume listing all.
        .END     INRING
```

Figure A–6. Listing of Program INRING.SR (concluded)

```
; This module defines the gate array. Generally,
;    the gate array must be defined in a separate
;    module. It must contain
;       .EXTG prog-entry-name
;    where prog-entry-name is the start entry name
;    in the program that will be accessed through
;    the gate. And it must be Linked with the inner
;    ring program; here:
;       X LINK/RING=5 INRING GATE.ARRAY
        .TITLE   GATE.ARRAY
        .EXTG    INRING    ; EXTGate defines gate
                           ;    for program INRING.
        ; Define 2-word pointer to gate array.
        .LOC     34        ; Locations 34 and 35.
        .DWORD   GATE.ARRAY  ; Pointer.
        .NREL 1            ; Shared code for general use.
        GATE.ARRAY: .DWORD  1   ; Gate array, 1 gate.
        .DWORD   (RING7-RING5)+INRING ; Address will be determined
                           ;    by Link.  Allows gate to be accessed
                           ;    by a program in any ring.
        RING7 = 7S3        ; Bits 1 3 specify gate 7.
        RING5 = 5S3        ; Bits 1 3 specify gate 5.
        .END
```

Figure A–7. Listing of Program GATE.ARRAY.SR

The assembly commands are

```
) XEQ MASM RINGLOAD ↵
) XEQ MASM INRING ↵
) XEQ MASM GATE.ARRAY ↵
```

The Link commands are

```
) XEQ LINK RINGLOAD ↵
) XEQ LINK/RING=5 INRING GATE.ARRAY ↵
```

The command to execute both programs is

```
) XEQ RINGLOAD ↵
```

The console output from the execution command is next.

I'm RINGLOAD, about to ?RINGLD program INRING.
I'm INRING, in inner ring. About to ?DEBUG. Type ESC R to proceed.

DEBUG 4 AOS/VS User Debugger Revision 006.000
34000001344 00000000000 12000000470 12001776206 00000000000
Warning- Memory Modification not allowed
_$R

From INRING — about to WRTN.

I'm RINGLOAD, back from INRING, terminating.

Program Set 4 — FILCREATE.SR

Program FILCREATE opens the terminal and asks you for the name of a file to create. If the file already exists, FILCREATE deletes and recreates it.

Figure A–8 contains FILCREATE.SR.

```
; This program opens the terminal and asks for a
;    filename. It then creates the file (deleting it
;    first if it exists) with an ACL of OWR. (An IPC
;    ?CREATE example occurs in programs SPEAK and HEAR.)
        .TITLE  FILCREATE
        .ENT    FILCREATE
        .NREL
FILCREATE: ?OPEN CONSOLE ; For input and output
        WBR     ERROR    ; Error, process it.
        ?WRITE  CONSOLE  ; Ask for filename.
        WBR     ERROR    ; Error, process it.
        XLEFB   0,BUF*2  ; Get buffer byte pointer.
        XWSTA   0,CONSOLE+?IBAD ; Put in I/O packet.
        ?READ   CONSOLE  ; Read filename.
        WBR     ERROR    ; Error, process it.
CREATE: ?CREATE CPKT     ; Create file (AC0 still has
                         ;    byte pointer to the name).
        WBR     TEST     ; Error, check for ERNAE.
        ?WRITE  CONSOLE  ; Done, confirm name.
        WBR     ERROR    ; Error, process it.
        XLEFB   0, TMES*2 ; Get confirmation message byte pointer.
        XWSTA   0,CONSOLE+?IBAD ; Put it in I/O packet.
        ?WRITE  CONSOLE  ; Write confirmation message.
        WBR     ERROR    ; Error, process it.
        WSUB    2,2      ; Good return flags.
        ?RETURN          ; Good bye.
        WBR     ERROR    ; Return error.
TEST:   WLDAI   ERNAE, 2 ; Code = 'already exists'.
        WSEQ    2,0      ; Skip if error is ERNAE.
        WBR     ERROR    ; Other error, process it.
        XLEFB   0, BUF*2 ; Get buffer byte pointer.
        ?DELETE          ; Delete the file.
        WBR     ERROR    ; Error, process it.
        WBR     CREATE   ; Go and create the file.
ERROR:  WLDAI   ?RFEC+?RFCF+?RFER,  2  ; Error flags.
        ?RETURN          ; To CLI.
        WBR     ERROR    ; Return error.
```

Figure A–8. Listing of Program FILCREATE.SR (continued)

```
        ; Create Packet.
        .ENABLE WORD      ; Most entries = 16 bits.
CPKT:   .BLK    ?CLTH     ; Packet length.
        .LOC    CPKT+?CFTYP
        ?ORDS*400+?FUDF   ; Data sensitive records, left byte
                          ;    (format mnemonic must always be *400)
                          ;    + User Data File.
        .LOC    CPKT+?CCPS
        0                 ; Fixed record length; ignored here.
        .LOC    CPKT+?CTIM
        .DWORD  -1        ; Default to current time.
        .LOC    CPKT+?CACP
        .DWORD  ACL*2     ; 2-word byte pointer to ACL.
        .LOC    CPKT+?CDEH
        0                 ; Reserved for system.
        .LOC    CPKT+?CDEL
        -1                ; Default element size.
        .LOC    CPKT+?CMIL
        -1                ; Default number of index levels.
        .LOC    CPKT+?CLTH ; Default others.
        ; End of Create packet.
ACL:    .TXT "DOCUMENT<0><?FACO+?FACW+?FACR><0>"
        ; Use your own username instead of DOCUMENT.
        ; Open and I/O packet -- needed for I/O.
CONSOLE: .BLK   ?IBLT     ; Packet length.
        .LOC    CONSOLE+?ISTI
        ?ICRF+?RTDS+?OFIO ; Data-sensitive records, Input/Output.
        .LOC    CONSOLE+?IMRS
         -1               ; Memory block size, default to 2048.
        .LOC    CONSOLE+?IBAD
        .DWORD  ITEXT*2   ; 2-word byte pointer to text/buffer.
        .LOC    CONSOLE+?IRCL
        120.              ; Maximum record length of 120 characters.
        .LOC    CONSOLE+?IFNP
        .DWORD  CON*2     ; 2-word byte pointer to filename.
        .LOC    CONSOLE+?IDEL
        .DWORD  -1        ; Data-sensitive delimiter table address;
                          ;    default to NL, CR, NUL.
        .LOC    CONSOLE+?IBLT ; Default others to 0.
        ; End of console I/O packet.
; Filename, buffer, messages. .NOLOC 1 follows.
        .NOLOC  1         ; To save listing space.
CON:    .TXT "@CONSOLE"   ; Use generic name.
ITEXT:  .TXT "Type valid filename to be created.  "
BUF:    .BLK    50.       ; Filename buffer, 100 chars.
TMES:   .TXT " created with ACL of OWR.<12>"
        .NOLOC  0         ; Resume listing all.
        .END    FILCREATE
```

Figure A–8. Listing of Program FILCREATE.SR (concluded)

The assembly, link, and execute commands are

```
) XEQ MASM FILCREATE ↵
) XEQ LINK FILCREATE ↵
) XEQ FILCREATE ↵
```

Typical lines of console dialog from the execution command are next.

```
) XEQ FILCREATE ↵
Type valid filename to be created.  FOO↵
FOO
 created with ACL of OWR.

) ACL/V FOO↵
FOO             DOCUMENT,OWR
```

Program Set 5 — WRITE.SR

Program WRITE opens the terminal and disk file FILE. It then asks you to type lines on the keyboard, and writes each line as a record into file FILE. When you type the three characters RD↵, program WRITE reads the records back from FILE, one at a time, and writes them on the terminal. WRITE includes two ?OPEN system calls (one with file deletion/creation) plus at least one ?READ, ?WRITE, and ?SPOS (for file positioning).

Figure A–9 contains WRITE.SR.

```
; This program opens the terminal (@CONSOLE), then
;    opens a disk file named FILE. This opening
;    creates FILE, deleting it first if it exists.
;    The program then reads input from the terminal,
;    a line at a time, and writes it to file FILE.
;    When the user types RD, the program positions
;    FILE at its beginning, writes all of its records
;    (lines) to the terminal, and ends. The example
;    shows calls ?OPEN, ?OPEN with file create/deletion,
;    ?READ, ?WRITE, ?SPOS, and ?CLOSE. There are
;    two ?OPEN packets: 1 for @CONSOLE, 1 for FILE.

        .TITLE  WRITE
        .ENT    WRITE
        .NREL   ; Default to partition 4.

        ; Open terminal (@CONSOLE) and FILE for input and output.

WRITE:  ?OPEN   CONSOLE  ; Open for Input/Output.
        WBR     ERROR    ; Error, process it.
        ?OPEN   FILE     ; Open (create?) FILE.
        WBR     ERROR    ; Error, process it.

        ; Write greeting and put I/O buffer byte pointer in the packet.

        ?WRITE  CONSOLE  ; Write message.
        WBR     ERROR    ; Error, process it.
        XLEFB   0,BUF*2  ; Get byte pointer to I/O buffer.
        XWSTA   0, CONSOLE+?IBAD ; Put in CON packet.

        ; Read line, check for terminator, write to FILE.

        NLDAI   'RD', 0  ; Put 'RD' terminator in AC0.

LOOP:   ?READ   CONSOLE  ; Read a line.
        WBR     ERROR    ; Error, process it.
        XNLDA   1, BUF   ; Get first word of buffer.
        WSNE    0,1      ; Did user type RD?
        WBR     SPOS     ;    Yes, do ?SPOS.
        ?WRITE  FILE     ;    No, write line to FILE.
        WBR     ERROR    ; Error, process it.
        WBR     LOOP     ; Get next line from user.

        ; Set position at beginning of FILE.

SPOS:   NLDAI   0, 1     ; Get 0 in AC1.
        XWSTA   1, FILE+?IRNH ; Put in record number word.
        XNLDA   2, FILE+?ISTI ; Get FILE's specifications.
        WMOV    2, 0          ; Save old specifications in AC0.
        WIORI   ?IPST,  2     ; Add ?IPST specification.
        XNSTA   2, FILE+?ISTI ; Put in FILE specifications.
        ?SPOS   FILE          ; Position at FILE start.
        WBR     ERROR         ; Error, process it.
        XNSTA   0, FILE+?ISTI ; Restore old specifications.
```

Figure A–9. Listing of Program WRITE.SR (continued)

```
        ; Read lines back from FILE, write to terminal.
LOOP1:  ?READ   FILE      ; Read from FILE into buffer.
        WBR     EOF       ; Error, test for EOF.
        ?WRITE  CONSOLE   ; Write line to CON.
        WBR     ERROR     ; Error, process it.
        WBR     LOOP1     ; Read/Write another line.
EOF:    NLDAI   EREOF, 2  ; Code for End of File.
        WSEQ    0, 2      ; Skip if error is EOF.
        WBR     ERROR     ; Other error, process.
        ; CLOSE the file.
CLOSE:  ?CLOSE  CONSOLE   ; Close terminal.
        WBR     ERROR     ; Error, process it.
        ?CLOSE  FILE      ; Close FILE.
        WBR     ERROR     ; Error, process it.
        WSUB    2,2       ; Set good return flags.
        WBR     BYE       ; Skip error flags.
        ;  Process error and/or return here.
ERROR:  WLDAI   ?RFEC+?RFCF+?RFER,  2  ; Error flags.
BYE:    ?RETURN           ; To CLI.
        WBR     ERROR     ; Return error.
        ; Open and I/O packet for terminal (minimum packet).
        .ENABLE WORD      ; Most entries = 16 bits.
        ; Open and I/O packet -- needed for I/O.
CONSOLE: .BLK   ?IBLT     ; Packet length.
        .LOC    CONSOLE+?ISTI
        ?ICRF+?RTDS+?OFIO ; Data-sensitive records, Input/Output.
        .LOC    CONSOLE+?IMRS
         -1               ; Memory block size, default to 2048.
        .LOC    CONSOLE+?IBAD
        .DWORD  ITEXT*2   ; 2-word byte pointer to message and
                          ;    later to I/O buffer.
        .LOC    CONSOLE+?IRCL
        120.              ; Maximum record length of 120 characters.
        .LOC    CONSOLE+?IFNP
        .DWORD  CON*2     ; 2-word byte pointer to filename.
        .LOC    CONSOLE+?IDEL
        .DWORD  -1        ; Data-sensitive delimiter table address;
                          ;    default to NL, CR, NUL.
        .LOC    CONSOLE+?IBLT ; Default others to 0.
        ; End of console I/O packet.
        ; Filename, buffer, messages.  .NOLOC 1 follows.
        .NOLOC  1         ; To save listing space.
CON:    .TXT "@CONSOLE"   ; Use generic name.
BUF:    .BLK 60.          ; 60. words, 120. characters.
ITEXT:  .TXT "I write lines to file FILE. Type
 RD[NL] to read lines back and stop.<12>"
        .NOLOC  0         ; Resume listing.
```

Figure A–9. Listing of Program WRITE.SR (continued)

```
        ; Open and I/O packet for FILE. You can omit
        ;    entries set to 0 if you specify a packet length
        ;    of ?IBLT, as shown with .BLK and .LOC here.
FILE:   .BLK    ?IBLT     ; Packet length.
        .LOC    FILE+?ICH
        0                 ; For system-created channel number.
        .LOC    FILE+?ISTI
        ?OFCR+?OFCE+?ICRF+?RTDS+?OFIO ; Delete and recreate,
                          ; change format if needed, data-sensitive
                          ; records, input and output.
        .LOC    FILE+?ISTO
        0                 ; Default file type.
        .LOC    FILE+?IMRS
         -1               ; Memory block size, default to 2048.
        .LOC    FILE+?IBAD
        .DWORD  BUF*2     ; Buffer. Use same as CONSOLE.
        .LOC    FILE+?IRES
        0                 ; Tape density, default to VSGEN.
        .LOC    FILE+?IRCL
        120.              ; Maximum record length of 120 characters.
        .LOC    FILE+?IRLR
        0                 ; System returns number of chars. transferred.
        .LOC    FILE+?IRNW
        0                 ; Reserved for system.
        .LOC    FILE+?IRNH
        .DWORD  0         ; 2 words for record number, default to
                          ;    the next record.
        .LOC    FILE+?IFNP
        .DWORD  FNAME*2   ; 2-word byte pointer to filename.
        .LOC    FILE+?IDEL
        .DWORD  -1        ; Data-sensitive delimiter table address;
                          ;    default to NL, CR, NUL.
        ; Default other specifications.
        .LOC    FILE+?IBLT ; Packet length.
        ; End of FILE packet.
FNAME:  .TXT  "FILE"      ; Disk filename.
        .END    WRITE
```

Figure A–9. Listing of Program WRITE.SR (concluded)

The assembly, link, and execute commands are

```
) XEQ MASM WRITE ↵
) XEQ LINK WRITE ↵
) XEQ WRITE ↵
```

Typical lines of dialog during the execution of program WRITE are next.

```
) XEQ WRITE ↵
I write lines to file FILE. Type RD[NL] to read lines back and stop.
LINE 1 OF 3. ↵
LINE 2 OF 3. ↵
LINE 3 (AND LAST) OF 3. ↵
RD ↵
LINE 1 OF 3.
LINE 2 OF 3.
LINE 3 (AND LAST) OF 3.

) TYPE FILE ↵
LINE 1 OF 3.
LINE 2 OF 3.
LINE 3 (AND LAST) OF 3.

)
```

Program Set 6 — DLIST.SR

Program DLIST gets a directory name from the CLI command line (?GTMES system call) and opens the directory and the line printer queue. Then, DLIST gets all filenames in the directory file and writes them to the line printer queue. The program includes I/O calls ?GNFN, ?OPEN, ?WRITE, and ?SEND.

Figure A–10 contains DLIST.SR.

```
; This program lists all filenames in a directory to the
;    line printer. It uses the ?GTMES system call to get the
;    directory name from the CLI command line and the system
;    call ?GOPEN to open the directory.  It shows system
;    calls ?GOPEN, ?OPEN, ?READ/?WRITE, ?GNFN, and ?SEND.
;    To execute it, type    X  prog-name  dir-name.
        .TITLE  DLIST
        .ENT    DLIST
        .NREL
        ; Get directory name, open directory file and the
        ;    line printer queue.
DLIST:  ?GTMES  CLIMSG    ; Get directory name argument.
        WBR     ERROR     ; Error, process it.
        LLEFB   0, DIRNAME*2 ; Get byte pointer to the directory name.
        NLDAI   -1, 1     ; -1 specifies system-assigned channel number
                          ;    for ?GOPEN.
        ?GOPEN  DIR       ; Open the directory.
        WBR     ERROR     ; Error, process it.
        ?OPEN   LINEP     ; Open @LPT queue.
        WBR     ERROR     ; Error, process it.
; Get next name via ?GNFN, write it to the printer.
        XNLDA   1, DIR+?ICH ; Keep channel number in AC1.
NEXT:   ?GNFN   GNAME     ; Put filename in ?GNFN buffer.
        WBR     EOF       ; Error, check for EOF.
        ?WRITE  LINEP     ; Write to LPT queue.
        WBR     ERROR     ; Error, process it.
        XLEFB   2, NL*2   ; Get address of NL character.
        XWSTA   2, LINEP+?IBAD ; Put it in LPT buffer.
        ?WRITE  LINEP     ; Write NL character to LPT queue.
        WBR     ERROR     ; Error, process it.
        XLEFB   2, FNAME*2  ; Get byte pointer to name buffer.
        XWSTA   2, LINEP+?IBAD ; Restore buffer address.
        WBR     NEXT      ; Get another filename.
EOF:    NLDAI   EREOF, 2  ; Code for End of File.
        WSEQ    0, 2      ; Skip if error was EOF.
        WBR     ERROR     ; Other error, process it.
```

Figure A–10. Listing of Program DLIST.SR (continued)

```
        ; Done with filenames. Get ?SEND parameters, issue ?SEND.
        XLEFB   0,CNAME*2 ; Byte pointer to console name.
        XLEFB   1,TMSG*2 ; Byte pointer to ?SEND message.
        WLDAI   (CLIMSG-TMSG)*2+1S22,  2  ; Message length +
                        ; byte pointer flag (1S22).
        ?SEND           ; Send message to @CONSOLE.
        WBR     ERROR   ; Error, process it.
        WSUB    2, 2    ; Done. Set good return flags.
        WBR     BYE     ; Goodbye.
ERROR:  NLDAI   ?RFEC+?RFCF+?RFER,  2  ; Error flags.
BYE:    ?RETURN                 ; To CLI.
        WBR     ERROR           ; Return error.
NL:     .TXT  "<12>"    ; Puts each name on new line.
        ; ?SEND console name and message.  .NOLOC 1 follows.
        .NOLOC  1
CNAME:  .TXT  "@CONSOLE" ; Use generic name.
TMSG:   .TXT  "All filenames written to LPT. Bye."
        .NOLOC  0       ; Resume listing all.
        .ENABLE WORD    ; Most packet entries = 16 bits.
        ; ?GTMES packet to get directory name from CLI command.
CLIMSG: .BLK    ?GTLN   ; ?GTMES packet length.
        .LOC    CLIMSG+?GREQ
        ?GARG   ; Put argument only in ?GRES.
        .LOC    CLIMSG+?GNUM
        1       ; Argument # 1 is directory name (0 is program name).
        .LOC    CLIMSG+?GRES
        .DWORD  DIRNAME*2 ; 2-word byte pointer to directory name buffer.
        .LOC    CLIMSG+?GTLN ; Default others to 0.
        ; End of ?GTMES packet.
DIRNAME: .BLK   50.     ; Buffer for directory pathname.
        ; ?OPEN packet, as needed for directory.
DIR:    .BLK    ?OPLT   ; ?GOPEN packet length.
                        ; No user specifications are needed
                        ;     for directory ?GOPEN.
        .LOC    DIR+?OPLT ; End of ?GOPEN packet.
        ; ?GNFN Packet to get next filename.
GNAME:  .BLK    ?NFLN   ; Packet length.
        .LOC    GNAME+?NFKY
        .DWORD  0       ; System uses the first 2 words.
        .LOC    GNAME+?NFNM
        .DWORD  FNAME*2  ; 2-word byte pointer to filename buffer.
        .LOC    GNAME+?NFTP
        .DWORD  -1      ; Template byte pointer, default, none.
        .LOC    GNAME+?NFLN ; End of ?GNFN pkt.
FNAME:  .BLK    16.     ; Maximum filename = 32 chars.
```

Figure A–10. Listing of Program DLIST.SR (continued)

```
        ; Open, I/O Packet for printer (output) file.
LINEP:  .BLK    ?IBLT     ; Packet length.
        .LOC    LINEP+?ICH
        0                 ; System puts channel number here.
        .LOC    LINEP+?ISTI
        ?ICRF+?RTDS+?OFOT ; Data-sensitive records, output.
        .LOC    LINEP+?ISTO
        0                 ; Default file type.
        .LOC    LINEP+?IMRS
         -1               ; Memory block size, default to 2048.
        .LOC    LINEP+?IBAD
        .DWORD  FNAME*2   ; 2-word byte pointer to text/buffer.
        .LOC    LINEP+?IRCL
        136.              ; Maximum record length of 136 characters.
        .LOC    LINEP+?IRLR
        0                 ; System returns number of chars transferred.
        .LOC    LINEP+?IRNH
        .DWORD  0         ; 2 words for record number; default to next.
        .LOC    LINEP+?IFNP
        .DWORD  LPTNM*2   ; 2-word byte pointer to filename.
        .LOC    LINEP+?IDEL
        .DWORD  -1        ; Data-sensitive delimiter table address;
                          ;    default to NL, CR, NUL.
        ; Other specifications can be 0.
        .LOC    LINEP+?IBLT ; Packet length.
         ; End of LINEP packet.
LPTNM:  .TXT  "@LPT"      ; Printer queue filename
                          ;    (could be @CONSOLE).
        .END    DLIST
```

Figure A–10. Listing of Program DLIST.SR (concluded)

The assembly, link, and execute commands are

```
) XEQ MASM DLIST ↵
) XEQ LINK DLIST ↵
) XEQ DLIST ↵
```

Output goes both to the line printer and to @CONSOLE. Typical line printer output is next in response to the CLI command

```
) XEQ DLIST = ↵
```

RUNTIME.PR
RINGLOAD.ST
INRING.SR
...
FILCREATE.OB

Typical @CONSOLE output is next in response to the CLI command

```
) XEQ DLIST = ↵
```

From Pid 49 : All filenames written to LPT. Bye.
)

Program Set 7 — NEWTASK.SR

In multitasked program NEWTASK, the program (initial) task creates another task, and then kills itself so that the newly created task receives exclusive control. NEWTASK issues system calls ?TASK, ?MYTID, and ?IDKIL.

Figure A–11 contains NEWTASK.SR.

```
; This program, NEWTASK, has the initial task
;    create a new task with priority 1 and ID 2.
;    The initial task opens the terminal, creates
;    the new task, announces its death, gets its
;    priority/ID, and kills itself. The new task
;    gets control, writes a message, and ?RETURNs.
;    (The last task cannot kill itself with a task
;    call.) The program shows calls ?TASK, ?MYTID,
;    and ?IDKIL.
        .TITLE  NEWTASK
        .ENT    NEWTASK
        .TSK    2           ; 2 tasks.
        .NREL               ; Unshared.
        ; Open terminal, create new task, kill self.
NEWTASK: ?OPEN  CONSOLE     ; For input and output.
        WBR     ERROR       ; Error, process it.
        ?TASK   TPACKET     ; Create new task,
                            ;    ID of 2, priority of 1.
        WBR     ERROR       ; Error, process it.
        ?WRITE  CONSOLE     ; Write swan song ...
        WBR     ERROR       ; Error, process it.
        ?MYTID              ; Get ID (AC0), priority (AC1).
        WBR     ERROR       ; Error, process it.
        WMOV    0, 1        ; Move ID into AC1 ...
        ?IDKIL              ;     ... and die.
        WBR     ERROR       ; Error on ?IDKIL.
        ; New task is now the only task.
NTASK:  XLEFB   0, NMSG*2 ; Get byte pointer to message.
        XWSTA   0, CONSOLE+?IBAD ; Put in I/O packet.
        ?WRITE  CONSOLE     ; Write message.
        WBR     ERROR       ; Error, process it.
        WSUB    2, 2        ; Set AC2 for good return.
        WBR     BYE         ; Go and return.
        ; Error handler.
ERROR:  NLDAI   ?RFEC+?RFCF+?RFER,  2  ; Error flags.
BYE:    ?RETURN             ; To CLI.
        WBR     ERROR       ; Return error.
```

Figure A–11. Listing of Program NEWTASK.SR (continued)

```
        ; Open and I/O packet for terminal.

        .ENABLE WORD ; Most entries = 16 bits.

CONSOLE: .BLK   ?IBLT     ; Packet length.
        .LOC    CONSOLE+?ISTI
        ?ICRF+?RTDS+?OFIO ; Data-sensitive records, I/O.
        .LOC    CONSOLE+?IMRS
        -1                ; Memory block size, default to 2048.
        .LOC    CONSOLE+?IBAD
        .DWORD  ITEXT*2   ; 2-word byte pointer to text/buffer.
        .LOC    CONSOLE+?IRCL
        120.              ; Maximum record length of 120 characters.
        .LOC    CONSOLE+?IFNP
        .DWORD  CON*2     ; 2-word byte pointer to filename.
        .LOC    CONSOLE+?IDEL
        .DWORD  -1        ; Data-sensitive delimiter table address; none.
        .LOC    CONSOLE+?IBLT ; End of I/O packet.

        ; Filename and messages.  .NOLOC 1 follows.

        .NOLOC  1
CON:    .TXT   "@CONSOLE"

ITEXT:  .TXT   "I'm the default task. Have opened
 @CONSOLE and am about to ?IDKIL myself.<12>"

NMSG:   .TXT   "I'm the new task, about to ?RETURN.<12>"
        .NOLOC  0         ; Resume listing all.

        ; ?TASK packet for new task.

TPACKET:  .BLK  ?DSLTH    ; Short packet length.
        .LOC    TPACKET+?DLNK
        1                 ; Any nonzero value for short packet.
        .LOC    TPACKET+?DLNL
        0                 ; Low order bits of ?DLNK, reserved.
        .LOC    TPACKET+?DLNKB
        .DWORD  0         ; ?DLNKB, ?DLNKBL are reserved.
        .LOC    TPACKET+?DPRI
        1                 ; Priority is 1 (initial task's
                          ;    priority is 0).
        .LOC    TPACKET+?DID
        2                 ; ID is 2 (initial task's ID is 1).
        .LOC    TPACKET+?DPC
        .DWORD NTASK      ; Task's starting address.
        .LOC    TPACKET+?DAC2
        .DWORD  0         ; No message for the new task.
        .LOC    TPACKET+?DSTB
        .DWORD  STACK     ; Stack address is STACK.
        .LOC    TPACKET+?DSFLT
        -1                ; Stack fault handler; default.
        .LOC    TPACKET+?DSSZ
        .DWORD  60.       ; Stack size is 60. words.
        .LOC    TPACKET+?DFLGS
        0                 ; Flag word, reserved.
        .LOC    TPACKET+?DRES
        0                 ; Reserved.
        .LOC    TPACKET+?DNUM
        1                 ; Number of tasks is 1.
        .LOC    TPACKET+?DSLTH ; End of ?TASK packet.

STACK:  .BLK    60.       ; 60. word stack for task.

        .END    NEWTASK
```

Figure A–11. Listing of Program NEWTASK.SR (concluded)

The assembly, link, and execute commands are

```
) XEQ MASM NEWTASK ↵
) XEQ LINK NEWTASK ↵
) XEQ NEWTASK ↵
```

The @CONSOLE output from the execution of program NEWTASK is next.

```
) XEQ NEWTASK ↵
I'm the default task. Have opened @CONSOLE and am about to ?IDKIL myself. I'm the new task,
about to ?RETURN.

)
```

Program Set 8 — BOOMER.SR

BOOMER is a fast multitasked (two tasks) file copy program. System calls ?XMTW and ?REC synchronize the program's read and write operations. BOOMER copies an existing input file to a freshly created output file. Either or both files can be tape or disk. You specify the filenames as arguments to the CLI command that executes program BOOMER.

Much of BOOMER is shared code to allow for general use. BOOMER illustrates system calls ?GTMES, ?TASK, ?XMTW, ?REC, ?KILL, ?OPEN, ?READ, and ?WRITE. BOOMER illustrates principles of input/output and of multitasking.

Figure A–12 contains BOOMER.SR.

```
; This program, BOOMER, has the initial task read from
;    an input file and an output task write to an output
;    file. There are two buffers. The input task reads
;    into BUFFER1 and transmits its address to the
;    output task. The output task then writes BUFFER1
;    while the input task fills BUFFER2. The output task
;    writes BUFFER2 and the cycle repeats. Both tasks
;    have priority 0; the output task has an ID of 2.
; For general use, all but packet and buffer code
;    is in shared area (.NREL 1). Each buffer is
;    8,192 bytes long, but you can change BUFLENGTH
;    to suit your needs.
; The program uses the CLI ?GTMES mechanism to
;    get the output and input filenames from the
;    command line. It assumes that the input file
;    exists.  It deletes/recreates the output file
;    with dynamic record format. Execute the program
;    with a CLI command of the form
;
;    X  progname  outputfile  inputfile
;
; Examples are:
;
;   X  BOOMER  @MTB0:3  :UDD:UTIL:PARU.32.SR
;
;   X  BOOMER  FOO  :UDD:UTIL:PARU.32.SR
        .TITLE  BOOMER
        .ENT    BOOMER
        .TSK    2       ; 2 tasks
        .NREL   1       ; Shared code.
; Initial task uses ?GTMES to get output filename
; (2nd argument), opens it; repeats ?GTMES to get input
; filename (1st argument), opens it; creates output task.
```

Figure A–12. Listing of Program BOOMER.SR (continued)

```
BOOMER: ?GTMES GPACKET   ; Get input filename.
        WBR     ERROR    ; Error, process it.
        LLEFB   0, FNAME*2  ; Get byte address of
                            ; ?GTMES-returned filename.
        LWSTA   0, INPUT+?IFNP ; Put in INPUT I/O packet.
        ?OPEN  INPUT     ; Open input file.
        WBR     ERROR    ; Error, process it.
        NLDAI   1, 0     ; Get 1 in AC0.
        LNSTA   0, GPACKET+?GNUM ; Specify arg1.
        ?GTMES  GPACKET  ; Get output filename.
        WBR     ERROR    ; Error, process it.
        LLEFB   0, FNAME*2 ; Get byte address of filename.
        LWSTA   0, OUTPUT+?IFNP ; Put in OUTPUT I/O packet.
        ?OPEN   OUTPUT   ; Open output file.
        WBR     ERROR    ; Error, process it.

        ?TASK   TPACKET  ; Create OUTPUT task.
        WBR     ERROR    ; Error, process it.

; Loop reads into BUFFER1, transmits it to output task,
;    reads into BUFFER2, transmits it to output task.
;    Message for output task is the buffer address.

READER: ?READ   INPUT    ; Read a buffer from input file.
        WBR     EOF?     ; Error, check for EOF.
        LLEF    0, MAILBOX  ; Get address of message.
        LWLDA   1, INPUT+?IBAD ; Message is buffer address.
        ?XMTW            ; Awaken output task.
        WBR     ERROR    ; Error, process it.

; Swap buffer byte pointers for next read.

        LLEFB   0, BUFFER1*2  ; Get byte pointer to BUFFER1.
        LLEFB   2, BUFFER2*2  ; Get byte pointer to BUFFER2.
        WSNE    1,2      ; Skip next instruction if BUFFER1
                         ;    was used for last READ.
        WMOV    0,2      ; Make BUFFER1 current buffer.
        LWSTA   2,INPUT+?IBAD ; Put current buffer byte pointer
                         ;    into INPUT packet.
        WBR     READER   ; Read into current buffer.

; On EOF, get number of characters read from INPUT packet
;    and make this number the buffer length for last ?XMTW.

EOF?:   NLDAI   EREOF, 1 ; Code for EOF (end of file).
        WSEQ    0, 1     ; Skip if not EOF.
        WBR     ERROR    ; Other error, process it.
        LLEF    0,MAILBOX ; Get address of message.
        LWLDA   1,INPUT+?IBAD  ; Message is buffer byte pointer.
        WMOV    1,2      ; Copy to AC2 for indexing.
        NLDAI   -1,3     ; Put -1 in AC3.
        WLSH    3,2      ; Make buffer byte pointer a word pointer.
        LNLDA   3,INPUT+?IRLR ; Get number of characters read
                         ;    from INPUT I/O packet.
        LWSTA   3,-2,2   ; Make BUFLENGTH (AC2-2) the number
                         ;    of characters read.
        ?XMTW            ; Send the last buffer.
        WBR     ERROR    ; Error, process it.
        ?KILL            ; Input is done; output task will ?RETURN.

; Error handler.

ERROR:  WLDAI   ?RFEC+?RFCF+?RFER,  2  ; Error flags.
        ?RETURN          ; To CLI.
        WBR     ERROR    ; Return error.
```

Figure A–12. Listing of Program BOOMER.SR (continued)

```
; Output task does the writing.
WRITER: LLEF    0,MAILBOX ; Get message address.
        ?REC              ; Wait for message.
        WBR     ERROR     ; Error, process it.
        LWSTA   1, OUTPUT+?IBAD  ; Got message.  It was a buffer byte
                                 ;    pointer; put it in I/O packet.
        WMOV    1,2       ; Copy to AC2 for indexing.
        NLDAI   -1,3      ; Put -1 in AC3.
        WLSH    3,2       ; Make byte pointer into word pointer.
        XWLDA   0,-2,2    ; Get buffer length left by input task
                          ;    (original length unless task hit EOF).
        LNSTA   0,OUTPUT+?IRCL ; Make this length the maximum record
                                ;    length in the I/O packet.
        ?WRITE  OUTPUT    ; Write buffer to output file.
        WBR     ERROR     ; Error, process it.
        WLDAI   BUFLENGTH, 1 ; Get original buffer length.
        WSNE    0,1       ; Is current buffer length the same as the
                          ;    original buffer length?
        WBR     WRITER    ;    Yes, get another buffer.
        WSUB    2,2       ;    No. Done. Set for good return.
        ?RETURN           ; Return to CLI.
        WBR     ERROR     ; Return error.

; Buffers, message, packets in unshared code.
        .NREL

; Buffer declarations.
BUFLENGTH =     16384.    ; Need change only this.
        BUFLENGTH         ; For residual characters after EOF.
BUFFER1: .BLK   (BUFLENGTH+1)/2   ; Size of BUFFER1.
        BUFLENGTH         ; For residual characters after EOF.
BUFFER2: .BLK   (BUFLENGTH+1)/2   ; Size of BUFFER2.

; Mailbox for message.
MAILBOX: 0
        .ENABLE WORD      ; Most packet items = 16 bits.

; ?GTMES packet to get Input and Output filenames.
GPACKET: .BLK   ?GTLN     ; ?GTMES packet length.
        .LOC    GPACKET+?GREQ
        ?GARG             ; Put argument only in ?GRES.
        .LOC    GPACKET+?GNUM
        2                 ; Argument 2 is input filename.
        .LOC    GPACKET+?GRES
        .DWORD  FNAME*2   ; 2-word byte pointer to filename buffer.
        .LOC    GPACKET+?GTLN ; Default others to 0.
        ; End of ?GTMES packet.
```

Figure A–12. Listing of Program BOOMER.SR (continued)

```
; Open and I/O packet for INPUT task.
INPUT:  .BLK    ?IBLT    ; Packet length.
        .LOC    INPUT+?ISTI
        ?ICRF+?RTDY+?OFIN ; Change format + Dynamic + Input.
        .LOC    INPUT+?IMRS
         -1               ; Memory block size, default to 2048.
        .LOC    INPUT+?IBAD
        .DWORD  BUFFER1*2 ;  Original input buffer.
        .LOC    INPUT+?IRCL
        BUFLENGTH        ; Record length is BUFLENGTH.
        .LOC    INPUT+?IRLR
        0                ; System returns number of chars. transferred.
        .LOC    INPUT+?IFNP
        .DWORD  FNAME*2  ; 2-word byte pointer to filename.
        .LOC    INPUT+?IDEL
        .DWORD  -1       ; Data-sensitive delimiter table address;
                         ;    not relevant.
        .LOC    INPUT+?IBLT ; Packet length.
        ; End of INPUT packet.
; ?TASK packet for output task - minimum packet.
TPACKET: .BLK   ?DSLTH   ; Short packet length.
        .LOC    TPACKET+?DLNK
        1                ; Any nonzero value for short packet.
        .LOC    TPACKET+?DPRI
        0       ; Priority is 0, as is input task's.
        .LOC    TPACKET+?DID
        2       ; ID is 2 (initial task's ID is 1).
        .LOC    TPACKET+?DPC
        .DWORD  WRITER   ; Task's starting address.
        .LOC    TPACKET+?DSTB
        .DWORD  STACK    ; Stack address is STACK.
        .LOC    TPACKET+?DSFLT
        -1               ; Stack fault handler; default.
        .LOC    TPACKET+?DSSZ
        .DWORD  60.      ; Stack size is 60. words.
        .LOC    TPACKET+?DNUM
        1                ; Number of tasks is 1.
        .LOC    TPACKET+?DSLTH ; End of ?TASK packet.
```

Figure A–12. Listing of Program BOOMER.SR (continued)

```
; Open and I/O packet for OUTPUT task.
OUTPUT: .BLK    ?IBLT    ; Packet length.
        .LOC    OUTPUT+?ISTI
        ?OFCR+?OFCE+?ICRF+?RTDY+?OFIO ; Delete + create, change format,
                                      ;    dynamic records, Input/Output.
        .LOC    OUTPUT+?IMRS
        -1              ; Memory block size, default to 2048.
        .LOC    OUTPUT+?IBAD
        .DWORD  BUFFER1*2 ; Start output from BUFFER1.
        .LOC    OUTPUT+?IRCL
        BUFLENGTH       ; Record length of BUFLENGTH chars.
        .LOC    OUTPUT+?IRLR
        0               ; System returns number of characters
                        ;    transferred here.
        .LOC    OUTPUT+?IFNP
        .DWORD  FNAME*2  ; 2-word byte pointer to filename.
        .LOC    OUTPUT+?IDEL
        .DWORD  -1      ; Data-sensitive delimiter table address;
                        ;    not relevant.
        .LOC    OUTPUT+?IBLT ; Default others to 0.
        ; End of Output I/O packet.
FNAME:  .BLK (?MXPL+1)/2 ; Filename  buffer; system limit for
                         ;    number of characters.
STACK:  .BLK    60.      ; 60.-word task stack.
        .END    BOOMER
```

Figure A–12. Listing of Program BOOMER.SR (concluded)

The assembly and link commands are

```
) XEQ MASM BOOMER ↵
) XEQ LINK BOOMER ↵
```

The execute command format is

```
XEQ BOOMER outputfilename inputfilename
```

A typical CLI command to execute BOOMER.PR is next. The command places a copy of (input) file MINE.DATA into (output) file YOURS.DATA. File YOURS.DATA has dynamic records.

```
) XEQ BOOMER YOURS.DATA MINE.DATA ↵

)
```

Program Set 9 — TIMEOUT.SR

TIMEOUT is a simple program that uses ?GTMES to get a number from the CLI command line that executes TIMEOUT, and then delays itself for this number of seconds. The number must be between 0 and 20., inclusive. Other examples of system call ?GTMES appear in programs DLIST.SR and BOOMER.SR in this appendix.

Figure A–13 contains TIMEOUT.SR.

```
; This program uses the CLI ?GTMES mechanism to get a number from
;    the CLI command that executed the program. Then it delays itself
;    for the given number of seconds. To run it, give the CLI command
;
;    X TIMEOUT secs
;
;    where    secs    is a decimal number from 0 through 20.

        .TITLE  TIMEOUT
        .ENT    TIMEOUT
        .NREL           ; Unshared.

; Use CLI ?GTMES to get number of secs. Puts
; ASCII value in AC2, binary value in AC1.

TIMEOUT: ?GTMES CLIMSG   ; Get number of seconds.
        WBR     ERROR    ; Error, process it.

; Check range of argument (returned in AC1).

        WCLM    1, 1     ; If not between values ...
        0                ;    ... lower limit of 0 ...
        20.              ;    ... upper limit of 20..
        WBR     BADVALUE ;    exit with "illegal" msg.

        NLDAI   1000., 0 ; Put 1000. in AC0.
        WMUL    1, 0     ; Get number of milliseconds in AC0.
        ?WDELAY          ; Delay for specified number of seconds.
        WBR     ERROR    ; Error, process it.
        WSUB    2, 2     ; Set for good return.
        WBR     BYE      ; Goodbye.

ERROR:  NLDAI ?RFEC+?RFCF+?RFER, 2   ; Error flags.
BYE:    ?RETURN                      ; To CLI.
        WBR     ERROR                ; Return error.

BADVALUE: XLEFB 1, BMSG*2            ; Byte pointer to message.
        NLDAI (CLIMSG-BMSG)*2+?RFCF, 2 ; Length + flags.
        WBR     BYE                  ; Bye.

BMSG:   .TXT  "You gave an illegal delay."

        ; ?GTMES packet to get number of seconds from the CLI command.

        .ENABLE WORD     ; Most entries = 16 bits.
CLIMSG: .BLK    ?GTLN    ; ?GTMES packet length.
        .LOC    CLIMSG+?GREQ
        ?GARG            ; Put argument only in ?GRES.
        .LOC    CLIMSG+?GNUM
        1                ; Argument 1 is number of seconds
                         ;    (Argument 0 is programname).
        .LOC    CLIMSG+?GRES
        .DWORD  -1       ; No buffer needed.
        .LOC    CLIMSG+?GTLN ; Default others; end packet.

        .END    TIMEOUT
```

Figure A–13. Listing of Program TIMEOUT.SR

The assembly, link, and execute commands are

```
) XEQ MASM TIMEOUT ↵
) XEQ LINK TIMEOUT ↵
) XEQ TIMEOUT ↵
```

Typical lines of dialog during the execution of program TIMEOUT are next.

```
) XEQ TIMEOUT 3

(3 seconds elapse.)

) XEQ TIMEOUT 22

You gave an illegal delay.

)
```

Program Set 10 — DIRCREATE.F77 and CHECK.F77

DIRCREATE.F77 is a FORTRAN 77 program that, at runtime, is a process that creates directory file NEW_DIR and makes it the process's working directory. The process issues system calls ?CREATE (which requires a parameter packet) and ?DIR (which does not require a parameter packet). The process also issues system call ?RETURN (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 the CLI program is a process named CLI.PR that accepts these two character strings, one at a time, and interprets them. Then, the CLI loads accumulators, creates any necessary parameter packets, and issues the respective system calls ?CREATE and ?DIR.

The point of this section is to list the general steps of writing FORTRAN 77 programs that issue explicit system calls. (All FORTRAN 77 programs issue implicit system calls; e.g., a TRUNCATE statement results in the runtime execution of system call ?TRUNCATE.) The section also contains a comprehensive example that you can easily generalize from. This section is a condensation of the chapter in the *FORTRAN 77 Environment Manual (AOS/VS)* that explains the system interface function, ISYS.

General Steps

The general steps are as follows:

1. List the system calls that your program issues and the PARU symbols that your program needs. The system call mnemonics, such as ?RETURN, are in file SYSID.32.SR. Place these symbols in a file whose name is based on the program name. An example for program MINE.F77 might be MINE_SYMBOLS, the contents of which are

   ```
   ?RETURN
   ?RFCF
   ?RFER
   ?RFEC
   ```

   Here, program MINE issues system call ?RETURN and uses PARU symbols ?RFCF, ?RFER, and ?RFEC. Make such a list for each source program unit that issues system calls.

2. Use program F77BUILD_SYM to read these symbols and create a file to include (via an %INCLUDE compiler directive or an INCLUDE statement) in your program. Do this for each source program unit that makes system calls. To continue the example in the previous step: F77BUILD_SYM reads symbol ?RFEC from file MINE_SYMBOLS and transforms it into the statements

```
INTEGER*2 ISYS_RFEC
PARAMETER (ISYS_RFEC = 4096)       ! ?RFEC = 10000K
```

   Although F77BUILD_SYM places the statements in file QSYM.F77.IN, you rename the file to MINE.F77.IN.

   F77BUILD_SYM also requires as input files SYSID.32.LS and PARU.32.LS. You use the Macroassembler to create these files.

   It's possible to build file QSYM.F77.IN with all the system call names in SYSID.32 and all the symbols in PARU.32. You could then have all your F77 programs that make explicit system calls include QSYM.F77.IN. This method eliminates the need for program–specific files such as MINE_SYMBOLS and MINE.F77.IN. However, it has its drawbacks. It results in several thousand extra statements (INTEGER and PARAMETER) for the compiler to process, larger object files, and larger program files.

3. Write your programs, being sure to have appropriate %INCLUDE compiler directives and INCLUDE statements. Compile, link, and test the resulting software.

Program Listings

Six related files appear next. Their figure numbers, names, and descriptions are as follows.

Figure Number	Filename	File Description
A–14	DIRCREATE.F77	Program to make system calls ?CREATE and ?DIR
A–15	DIRCREATE_SYMBOLS	Symbols in files SYSID.32 and PARU.32
A–16	DIRCREATE_SYMBOLS.F77.IN	Symbols and values from SYSID.32 and PARU.32; included in DIRCREATE.F77
A–17	CHECK.F77	Subroutine to report errors that occurred during a system call
A–18	CHECK_SYMBOLS	Symbols in files SYSID.32 and PARU.32
A–19	CHECK_SYMBOLS.F77.IN	Symbols and values from SYSID.32 and PARU.32; included in CHECK.F77

Figure A–14 contains program DIRCREATE.F77.

```
      PROGRAM DIRCREATE
C     This program is a complete example program that makes
C        system calls ?CREATE (with a parameter packet) and
```

```
C        ?DIR (without a parameter packet).
      INTEGER*4 AC0, AC1, AC2 ! Accumulators
      CHARACTER*8 NEW_DIR / 'NEW_DIR<0>' /
      INTEGER*4 RESULT_CODE   ! Result of system call
C     Define the parameter packet for ?CREATE.
      INTEGER*2 CREATE_PACKET(0:9)      ! Ten 16-bit words
      INTEGER*2 ?CFTYP          ! offset ?CFTYP, single word
      EQUIVALENCE (CREATE_PACKET(0),   ?CFTYP)
      INTEGER*2 ?CHFS           ! offset ?CHFS,  single word
      EQUIVALENCE (CREATE_PACKET(1),   ?CHFS)
      INTEGER*4 ?CTIM           ! offset ?CTIM,  double word
      EQUIVALENCE (CREATE_PACKET(2),   ?CTIM)
      INTEGER*4 ?CACP           ! offset ?CACP,  double word
      EQUIVALENCE (CREATE_PACKET(4),   ?CACP)
      INTEGER*4 ?CMSH           ! offset ?CMSH,  double word
      EQUIVALENCE (CREATE_PACKET(6),   ?CMSH)
      INTEGER*2 ?CMIL           ! offset ?CMIL,  single word
      EQUIVALENCE (CREATE_PACKET(10K), ?CMIL)
C or  EQUIVALENCE (CREATE_PACKET( 8 ), ?CMIL)
      INTEGER*2 ?CMRS           ! offset ?CMRS,  single word
      EQUIVALENCE (CREATE_PACKET(11K), ?CMRS)
C or  EQUIVALENCE (CREATE_PACKET( 9 ), ?CMRS)
C     File DIRCREATE_SYMBOLS.F77.IN contains, from files SYSID.32
C        and PARU.32, symbols and values for this program.
      INCLUDE 'DIRCREATE_SYMBOLS.F77.IN'
C     Load AC0 with the byte address of the new directory's name.
      AC0 = BYTEADDR(NEW_DIR)
      AC1 = 0
C     Prepare the parameter packet for ?CREATE.
      ?CFTYP = 0*400K + ISYS_FDIR  ! Left byte is 0, right byte is ?FDIR
      ?CHFS = -1
      ?CTIM = -1
      ?CACP = -1
      ?CMSH = 0
      ?CMIL = -1
      ?CMRS = 0
      AC2 = WORDADDR(CREATE_PACKET)
C     Make system call ?CREATE, according to accumulator values
C        and its parameter packet, via function subprogram ISYS.
      RESULT_CODE = ISYS(ISYS_CREATE, AC0, AC1, AC2)
C     Report any errors that occurred during ?CREATE.
      CALL CHECK (RESULT_CODE, 'While creating directory NEW_DIR')
C     Set up accumulators for system call ?DIR.
      AC0 = BYTEADDR(NEW_DIR)
      AC1 = 0
      AC2 = 0
C     Make system call ?DIR, according to accumulator values,
C        via function subprogram ISYS.
      RESULT_CODE = ISYS(ISYS_DIR, AC0, AC1, AC2)
C     Report any errors that occurred during ?DIR.
      CALL CHECK (RESULT_CODE, 'While moving to directory NEW_DIR')
      STOP
      END
```

Figure A–14. Listing of Program DIRCREATE.F77

Figure A–15 contains file DIRCREATE_SYMBOLS.

```
?CREATE
?DIR
?FDIR
```

Figure A–15. File DIRCREATE_SYMBOLS

Figure A–16 contains file DIRCREATE_SYMBOLS.F77.IN. Program F77BUILD_SYM creates QSYM.F77.IN from file DIRCREATE_SYMBOLS; you rename QSYM.F77.IN to DIRCREATE_SYMBOLS.F77.IN.

```
****  F77 INCLUDE file for system parameters ****
****  INTEGER*4 parameters for SYSID  ****
      INTEGER*4  ISYS_CREATE
      PARAMETER (ISYS_CREATE = 0)    ! ?.CREATE = 0K
      INTEGER*4  ISYS_DIR
      PARAMETER (ISYS_DIR = 61)      ! ?.DIR = 75K
****  Parameters for PARU   ****
      INTEGER*2  ISYS_FDIR
      PARAMETER (ISYS_FDIR = 10)     ! ?FDIR = 12K
****  END of F77 INCLUDE file for system parameters ****
```

Figure A–16. File DIRCREATE_SYMBOLS.F77.IN

Suppose you give the CLI commands

```
) DELETE/2=IGNORE DIRCREATE.LS ↵
) F77/L=DIRCREATE.LS DIRCREATE.F77 ↵
) QPRINT DIRCREATE.LS ↵
```

Then, the contents of DIRCREATE_SYMBOLS.F77.IN appear on paper just after the statement

```
INCLUDE 'DIRCREATE_SYMBOLS.F77.IN'
```

Figure A–17 contains subroutine subprogram CHECK.F77. This is a general–purpose routine that you can call from any program that invokes the system interface function ISYS. Suppose CHECK receives an error code of 0 in its parameter ECODE. CHECK simply returns, because an error code of 0 means that the system call executed without an error. If CHECK receives a nonzero error code, it responds by executing ?RETURN and displaying whatever message CHECK has received in its parameter TEXT. You might want to change CHECK's response to a nonzero error code to meet your installation's needs.

```
      SUBROUTINE CHECK (ECODE, TEXT)
      INTEGER*4 ECODE          ! Error code from main program,
                               !   returned there from ISYS
      CHARACTER*(*) TEXT       ! Error text from main program
                               !   to accompany ECODE
      INTEGER*4 AC2            ! Accumulator 2
      INTEGER*4 RESULT         ! of ?RETURN call
      INCLUDE 'CHECK_SYMBOLS.F77.IN'
      IF (ECODE .EQ. 0) THEN
            RETURN             ! ISYS executed without an error.
      END IF
C     ISYS executed with an error, so report it.
      AC2 =    ISYS_RFCF       ! CLI format
     *       + ISYS_RFER       ! Error -- Severity = 2
     *       + ISYS_RFEC       ! AC0 contains the error code
      AC2 = AC2 + MIN(LEN(TEXT),255)
C     Execute ?RETURN and report the error from ISYS.
      RESULT = ISYS(ISYS_RETURN, ECODE, BYTEADDR(TEXT), AC2)
      STOP '- Impossible-to-occur error occurred during ?RETURN'
      END
```

Figure A–17. Listing of Subroutine CHECK.F77

Figure A–18 contains file CHECK_SYMBOLS.

```
?RETURN
?RFCF
?RFER
?RFEC
```

Figure A–18. File CHECK_SYMBOLS

Figure A–19 contains file CHECK_SYMBOLS.F77.IN. Program F77BUILD_SYM creates QSYM.F77.IN from file CHECK_SYMBOLS; you rename QSYM.F77.IN to CHECK_SYMBOLS.F77.IN.

```
****  F77 INCLUDE file for system parameters ****
****  INTEGER*4 parameters for SYSID  ****
      INTEGER*4  ISYS_RETURN
      PARAMETER (ISYS_RETURN = 200)    ! ?.RETURN = 310K
****  Parameters for PARU   ****
      INTEGER*2  ISYS_RFCF
      PARAMETER (ISYS_RFCF = -32768)   ! ?RFCF = 100000K
      INTEGER*2  ISYS_RFEC
      PARAMETER (ISYS_RFEC = 4096)     ! ?RFEC = 10000K
      INTEGER*2  ISYS_RFER
      PARAMETER (ISYS_RFER = 16384)    ! ?RFER = 40000K
****  END of F77 INCLUDE file for system parameters ****
```

Figure A–19. File CHECK_SYMBOLS.F77.IN

Suppose you give the CLI commands

```
) DELETE/2=IGNORE CHECK.LS ⤶
) F77/L=CHECK.LS CHECK.F77 ⤶
) QPRINT CHECK.LS ⤶
```

Then, the contents of CHECK_SYMBOLS.F77.IN are printed just after the statement

```
INCLUDE 'CHECK_SYMBOLS.F77.IN'
```

Program Construction

First, be sure you have at least Read access to files SYSID.32.LS and PARU.32.LS. Program F77BUILD_SYM, to which you must have at least Execute access, requires these files. Typically, your system manager creates them in :UTIL via executing the Macroassembler. The *FORTRAN 77 Environment Manual (AOS/VS)* explains how to create these listing files.

Next is dialog that shows the construction of program file DIRCREATE.PR from files DIRCREATE.F77 (Figure A–14), DIRCREATE_SYMBOLS (Figure A–15), DIRCREATE_SYMBOLS.F77.IN (Figure A–16), CHECK.F77 (Figure A–17), CHECK_SYMBOLS (Figure A–18), and CHECK_SYMBOLS.F77.IN (Figure A–19). All filename suffixes appear for emphasis; you frequently can ignore them during commands to the compiler and Link. The output from the compiler and Link programs doesn't appear in the dialog.

```
) COMMENT Create QSYM.F77.IN from DIRCREATE_SYMBOLS. ↵
) XEQ F77BUILD_SYM DIRCREATE_SYMBOLS ↵
) DELETE/2=IGNORE DIRCREATE_SYMBOLS.F77.IN ↵
) RENAME QSYM.F77.IN DIRCREATE_SYMBOLS.F77.IN ↵
) COMMENT DIRCREATE_SYMBOLS.F77.IN is ready for inclusion ↵
) COMMENT   by DIRCREATE.F77. ↵
) COMMENT Create DIRCREATE.OB from DIRCREATE.F77 and ↵
) COMMENT   DIRCREATE_SYMBOLS.F77.IN. ↵
) F77 DIRCREATE.F77 ↵
) COMMENT ↵
) COMMENT Create QSYM.F77.IN from CHECK_SYMBOLS. ↵
) XEQ F77BUILD_SYM CHECK_SYMBOLS ↵
) DELETE/2=IGNORE CHECK_SYMBOLS.F77.IN ↵
) RENAME QSYM.F77.IN CHECK_SYMBOLS.F77.IN ↵
) COMMENT CHECK_SYMBOLS.F77.IN is ready for inclusion by CHECK.F77. ↵
) COMMENT Create CHECK.OB from CHECK.F77 and CHECK_SYMBOLS.F77.IN. ↵
) F77 CHECK.F77 ↵
) COMMENT CREATE DIRCREATE.PR from DIRCREATE.OB and CHECK.OB ↵
) F77LINK DIRCREATE.OB CHECK.OB ↵
) COMMENT Program file DIRCREATE.PR is now ready for execution. ↵
```

Program Execution

Execute FORTRAN 77 programs that make explicit system calls in the same way that you execute any other FORTRAN 77 program.

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

```
) COMMENT EXECUTE DIRCREATE.PR after making sure that NEW_DIR ↵
) COMMENT   doesn't exist. ↵
) DELETE/2=IGNORE NEW_DIR ↵
) DIRECTORY ↵

:UDD4:ALICE
```

```
) DATE; TIME ↵

26-AUG-85
16:38:37

) XEQ DIRCREATE.PR ↵

STOP

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

DIRECTORY :UDD4:ALICE

 NEW_DIR        DIR  26-AUG-85  16:38:44      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 again, ↵
) COMMENT   and see what happens when ?CREATE tries to create a file ↵
) COMMENT   (NEW_DIR) that already exists. ↵
) XEQ DIRCREATE.PR ↵

*ERROR*
FILE NAME ALREADY EXISTS
While creating directory NEW_DIR
ERROR: FROM PROGRAM
XEQ,DIRCREATE.PR

) COMMENT Notice how subroutine CHECK reported the error and ↵
) COMMENT   terminated the process. ↵
```

Program Set 11 — CREATE_WINDOW.SR

Program CREATE_WINDOW uses system call ?WINDOW to create a character window. After creating and setting up the window, the program uses system call ?PROC to pass the window to a CLI son process. After the CLI process terminates, CREATE_WINDOW deletes the window.

Figure A–20 contains CREATE_WINDOW.SR.

```
        .TITLE CREATE_WINDOW
        .ENT CREATE_WINDOW
        .NREL 0
; This program creates a character window, gets its status, adjusts its
;   appearance, and makes the window visible to the user.  The program
;   then issues ?PROC to create a CLI process into the window and pends
;   until the user terminates the CLI.  The program then deletes the window.
; The program assumes that its @CONSOLE file is a window on a pixel map.
CREATE_WINDOW:
        ; Create a character window using ?WINDOW system call,
        ; function code ?WIN_CREATE_WINDOW.
CREATE:
        XLEF    2,WIN_PKT                       ; Load word pointer to main
                                                ;   packet into AC2.
        NLDAI   ?WIN_CREATE_WINDOW,0            ; Specify ?WIN_CREATE_WINDOW
        XNSTA   0,?WIN_PKT.FUNC,2               ;   function code in the
                                                ;   main packet.
        WLDAI   1S?WIN_PKT.FLAGS.IN_PATH,0      ; Specify window pathname
        XWSTA   0,?WIN_PKT.FLAGS,2              ;   by setting flag in the
                                                ;   main packet.
        XLEFB   0,PMAP*2                        ; Store byte pointer to
                                                ;   pathname in the
        XWSTA   0,?WIN_PKT.PATH,2               ;   main packet.
        NLDAI   PMAP_PATH_LENGTH,0              ; Store length of pathname
        XNSTA   0,?WIN_PKT.PATH_BUF_LEN,2       ;   in the main packet.
        XLEF    0,WIN_CRE                       ; Store word pointer to
        XWSTA   0,?WIN_PKT.SUBPKT,2             ;   subpacket in main packet.
        WSUB    0,0                             ; AC0 must be set to zero.
        WSUB    1,1                             ; AC1 must be set to zero.
        ?WINDOW                                 ; Create a window.
        WBR     ERROR                           ; Quit.
        ; Use window ID as window specifier for remaining ?WINDOW calls.
        XWLDA   0,?WIN_PKT.WIND_ID,2            ; Load window ID and store
        XWSTA   0,WIND_ID                       ;   it in location WIND_ID.
        WLDAI   1S?WIN_PKT.FLAGS.IN_WIND_ID,0   ; Use window ID as specifier
        XWSTA   0,?WIN_PKT.FLAGS,2              ;   by setting flag in the
                                                ;   main packet.
        WSUB    0,0                             ; Clear AC0.
        XWSTA   0,?WIN_PKT.PATH,2               ; Clear pathname in the main
                                                ;   packet.
        XNSTA   0,?WIN_PKT.PATH_BUF_LEN,2       ; Clear pathname length.
```

Figure A–20. Listing of Program CREATE_WINDOW.SR (continued)

```
        ; Get window status information using function code
        ;   ?WIN_WINDOW_STATUS.
STATUS:
        XLEF    2,WIN_PKT                       ; Load word pointer to main
                                                ;   packet in AC2.
        NLDAI   ?WIN_WINDOW_STATUS,0            ; Specify ?WIN_WINDOW_STATUS
        XNSTA   0,?WIN_PKT.FUNC,2               ;   function code in the
                                                ;   main packet.
        XLEF    0,WIN_STAT                      ; Store word pointer to the
                                                ;   subpacket in the main
        XWSTA   0,?WIN_PKT.SUBPKT,2             ;   packet.
        WSUB    0,0                             ; Clear AC0.
        WSUB    1,1                             ; Clear AC1.
        ?WINDOW                                 ; Get status. System returns
                                                ;   information in subpacket.
        WBR     ERROR                           ; Quit.
        ; Adjust the appearance of the window using the ?WINDOW system
        ;   call, function code ?WIN_DEFINE_PORTS.
DEF_PORTS:
        XLEF    2,WIN_PKT                       ; Load word pointer to main
                                                ;   packet in AC0.
        NLDAI   ?WIN_DEFINE_PORTS,0             ; Specify ?WIN_DEFINE_PORTS
        XNSTA   0,?WIN_PKT.FUNC,2               ;   function code in the
                                                ;   main packet.
        XLEF    0,WIN_DPORT                     ; Store word pointer to
                                                ;   subpacket in the main
        XWSTA   0,?WIN_PKT.SUBPKT,2             ;   packet.
        WSUB    0,0                             ; Clear AC0.
        WSUB    1,1                             ; Clear AC1.
        ?WINDOW                                 ; Define view port and
                                                ;   scan port.
        WBR     ERROR                           ; Quit.
        ; Make the window visible to the user using the ?WINDOW system
        ;   call, function code ?WIN_UNHIDE_WINDOW.
UNHIDE:
        XLEF    2,WIN_PKT                       ; Load word pointer to main
                                                ;   packet in AC0.
        NLDAI   ?WIN_UNHIDE_WINDOW,0            ; Specify ?WIN_UNHIDE_WINDOW
        XNSTA   0,?WIN_PKT.FUNC,2               ;   function code in the
                                                ;   main packet.
        WSUB    0,0                             ; Clear AC0.
        WSUB    1,1                             ; Clear AC1.
        XWSTA   0,?WIN_PKT.SUBPKT,2             ; No subpacket for this
                                                ;   function.
        ?WINDOW                                 ; Unhide the window.
        WBR     ERROR                           ; Quit.
```

Figure A-20. Listing of Program CREATE_WINDOW.SR (continued)

```
        ; ?PROC a CLI, passing the window as the CLI's @CONSOLE
        ;   file.  AOS/VS assigns the window to the CLI.
PROC_CLI:
        WSUB    0,0                             ; Clear AC0.
        WSUB    1,1                             ; Clear AC1.
        XLEF    2,PROC_PKT                      ; Load word pointer to the
                                                ;   ?PROC packet in AC0.
        ?PROC                                   ; ?PROC a CLI in the
                                                ;   window.
        WBR     ERROR                           ; Quit.

        ; The program pends here until the user terminates the CLI.
        ;   Unless there is an error, the program then deletes the
        ;   window.

        ; Delete the window using the ?WINDOW system call, function code
        ;   ?WIN_DELETE_WINDOW.
DELETE:
        XLEF    2,WIN_PKT                       ; Load word pointer to the
                                                ;   main packet in AC0.
        NLDAI   ?WIN_DELETE_WINDOW,0            ; Specify ?WIN_DELETE_WINDOW
        XNSTA   0,?WIN_PKT.FUNC,2               ;   function code in the
                                                ;   main packet.
        WSUB    0,0                             ; Clear AC0.
        WSUB    1,1                             ; Clear AC1.
        XWSTA   0,?WIN_PKT.SUBPKT,2             ; No subpacket for this
                                                ;   function.
        ?WINDOW                                 ; Delete the window.
        WBR     ERROR                           ; Quit.

        WSUB    2,2                             ; Set flags for a normal
                                                ;   return.
        WBR     BYE                             ; Branch to good return.

        ; Return to the calling process.
ERROR:  WLDAI   ?RFEC!?RFCF!?RFER,2             ; Error flags:
                                                ; Error code is in AC0
                                                ;   (?RFEC), message is in
                                                ;   CLI format (?RFCF),
                                                ;   caller should handle
                                                ;   this as an error (?RFER).
BYE:    ?RETURN                                 ; Return to CLI.
        WBR     ERROR                           ; ?RETURN error return.
```

Figure A–20. Listing of Program CREATE_WINDOW.SR (continued)

```
        ; Set up a place to save the window ID.
WIND_ID: 0
        ; Set up the main ?WINDOW packet
WIN_PKT: .BLK   ?WIN_PKT_LEN                    ; Allocate enough space for
                                                ;   the ?WINDOW packet.
        .LOC    WIN_PKT+?WIN_PKT.PKT_ID         ; Packet ID.
        .DWORD  ?WIN_PKT_PKTID
        .LOC    WIN_PKT+?WIN_PKT.FUNC           ; Function code.
        .WORD   0
        .LOC    WIN_PKT+?WIN_PKT.CHAN_NUM       ; Channel number of
                                                ;   target window.
        .WORD   0
        .LOC    WIN_PKT+?WIN_PKT.FLAGS          ; Flags: window specifier.
        .DWORD  0
        .LOC    WIN_PKT+?WIN_PKT.PATH           ; Pathname of target window.
        .DWORD  0
        .LOC    WIN_PKT+?WIN_PKT.PATH_BUF_LEN   ; Length of pathname
        .WORD   0                               ;  including null terminator.
        .LOC    WIN_PKT+?WIN_PKT.PATH_LEN       ; Length of pathname returned
        .WORD   0                               ;   by AOS/VS.
        .LOC    WIN_PKT+?WIN_PKT.WIND_ID        ; Window ID of target window.
        .DWORD  0
        .LOC    WIN_PKT+?WIN_PKT.SUBPKT         ; Word pointer to function
        .DWORD  0                               ;   subpacket or zero.
        .LOC    WIN_PKT+?WIN_PKT_LEN            ; End of packet.
        ; Set up storage for some ?WIN_CREATE parameters.
        PMAP:           .TXT "@CONSOLE"
        PMAP_PATH_LENGTH = 9.
        WINDOW_NAME:   .TXT "NEW_WINDOW"
        NAME_LEN = 11.
        WINDOW_TITLE: .TXT "CLI"
        TITLE_LEN = 4.
        WINDOW_PATH:   .TXT "@CONSOLE:NEW_WINDOW"
        ; Define some masks for the ?WIN_CREATE flags word.
        NEW_GRP_MSK = 1S?WIN_CRE.FLAGS.NEW_GRP
        VT_SZ_X_MSK = 1S?WIN_CRE.FLAGS.VT_SZ_X
        VT_SZ_Y_MSK = 1S?WIN_CRE.FLAGS.VT_SZ_Y
        CRE_SON_MSK = 1S?WIN_CRE.FLAGS.CREATE_SON
```

Figure A–20. Listing of Program CREATE_WINDOW.SR (continued)

```
        ; Set up the ?WIN_CREATE subpacket.
WIN_CRE: .BLK    ?WIN_CRE_LEN_REV1              ; Allocate enough space for
                                                ;   ?WIN_CREATE subpacket.
        .LOC     WIN_CRE+?WIN_CRE.PKT_ID        ; Packet ID.
        .DWORD   ?WIN_CRE_PKTID_REV1
        .LOC     WIN_CRE+?WIN_CRE.FLAGS         ; Flags: the window starts a
                                                ;   new group, and the width
                                                ;   and height of the virtual
                                                ;   terminal are specified
                                                ;   with this call.
        .DWORD   NEW_GRP_MSK!VT_SZ_X_MSK!VT_SZ_Y_MSK
        .LOC     WIN_CRE+?WIN_CRE.VT_TYPE       ; Virtual terminal type:
        .DWORD   ?WIN_VT_TYPE_D460              ;   D460 character window.
        .LOC     WIN_CRE+?WIN_CRE.VT_SZ.X       ; Width of virtual terminal
        .DWORD   162.                           ;   is 162. (character)
                                                ;   columns.
        .LOC     WIN_CRE+?WIN_CRE.VT_SZ.Y       ; Height of virtual terminal
        .DWORD   24.                            ;   is 24. (character) rows.
        .LOC     WIN_CRE+?WIN_CRE.TITLE         ; Window title:
        .DWORD   WINDOW_TITLE*2                 ;   byte pointer to
                                                ;   WINDOW_TITLE.
        .LOC     WIN_CRE+?WIN_CRE.TITLE_BUF_LEN ; Length of window title,
        .WORD    TITLE_LEN                      ;   including a null
                                                ;   terminator.
        .LOC     WIN_CRE+?WIN_CRE.PIXEL_DEPTH   ; Bits per pixel for pixel
        .WORD    0                              ;   map: not valid for
                                                ;   character window.
        .LOC     WIN_CRE+?WIN_CRE.INPUT_BUF_SZ  ; Length of input buffer is
        .WORD    0                              ;   the default buffer size.
        .LOC     WIN_CRE+?WIN_CRE.WNAME_BUF_LEN ; Length of window name,
        .WORD    name_len                       ;   including a null
                                                ;   terminator.
        .LOC     WIN_CRE+?WIN_CRE.WNAME         ; Window name:
        .DWORD   WINDOW_NAME*2                  ;   byte pointer to
                                                ;   WINDOW_NAME.
        .LOC     WIN_CRE+?WIN_CRE.PAL_TYPE      ; Palette type:
        .DWORD   ?WIN_PALETTE_TYPE_D460         ;   use D460 palette.
        .LOC     WIN_CRE+?WIN_CRE.BORDER_TYPE   ; Border type: specify
        .WORD   ?WIN_BORDER_TYPE_TITLE          ;   a title border type.
        .LOC     WIN_CRE+?WIN_CRE.RSV           ; Reserved value must be
        .WORD    0                              ;   set to zero.
        .LOC     WIN_CRE+?WIN_CRE.VIEW_ORG.X    ; X-coordinate of view port
        .DWORD   0                              ;   origin: set later with
                                                ;   ?DEFINE_PORTS.
        .LOC     WIN_CRE+?WIN_CRE.VIEW_ORG.Y    ; Y-coordinate of view port
        .DWORD   0                              ;   origin: set later with
                                                ;   ?DEFINE_PORTS.
        .LOC     WIN_CRE+?WIN_CRE_LEN_REV1      ; End of subpacket.
```

Figure A–20. Listing of Program CREATE_WINDOW.SR (continued)

```
        ; Set up the ?WIN_WINDOW_STATUS subpacket
WIN_STAT: .BLK  ?WIN_STATUS_LEN_REV1            ; Allocate enough space for
                                                ;   ?WIN_WINDOW_STATUS
                                                ;   subpacket.
        .LOC    WIN_STAT+?WIN_STATUS.PKT_ID     ; Packet ID.
        .DWORD  ?WIN_STATUS_PKTID_REV1
        .LOC    WIN_STAT+?WIN_STATUS.FLAGS      ; Window state flags.
        .DWORD  0
        .LOC    WIN_STAT+?WIN_STATUS.VIEW_ORG.X ; X-coordinate of view
        .DWORD  0                               ;   port origin (pixels).
        .LOC    WIN_STAT+?WIN_STATUS.VIEW_ORG.Y ; Y-coordinate of view
        .DWORD  0                               ;   port origin (pixels).
        .LOC    WIN_STAT+?WIN_STATUS.VIEW_SZ.X  ; Width of view port on
        .DWORD  0                               ;   physical screen
                                                ;   (pixels).
        .LOC    WIN_STAT+?WIN_STATUS.VIEW_SZ.Y  ; Height of view port on
        .DWORD  0                               ;   physical screen
                                                ;   (pixels).
        .LOC    WIN_STAT+?WIN_STATUS.SCAN_ORG.X ; X-coordinate of scan
        .DWORD  0                               ;   port origin (chars).
        .LOC    WIN_STAT+?WIN_STATUS.SCAN_ORG.Y ; Y-coordinate of scan
        .DWORD  0                               ;   port origin (chars).
        .LOC    WIN_STAT+?WIN_STATUS.SCAN_SZ.X  ; Width of scan port on
        .DWORD  0                               ;   virtual screen (chars).
        .LOC    WIN_STAT+?WIN_STATUS.SCAN_SZ.Y  ; Height of scan port on
        .DWORD  0                               ;   virtual screen (chars).
        .LOC    WIN_STAT+?WIN_STATUS.WIND_ORG.X ; X-coordinate of window
        .DWORD  0                               ;   origin (pixels).
        .LOC    WIN_STAT+?WIN_STATUS.WIND_ORG.Y ; Y-coordinate of window
        .DWORD  0                               ;   origin (pixels).
        .LOC    WIN_STAT+?WIN_STATUS.WIND_SZ.X  ; Width of window and
        .DWORD  0                               ;   border (pixels).
        .LOC    WIN_STAT+?WIN_STATUS.WIND_SZ.Y  ; Height of window and
        .DWORD  0                               ;   border (pixels).
        .LOC    WIN_STAT+?WIN_STATUS.VT_SZ.X    ; Width of virtual
        .DWORD  0                               ;   terminal (chars).
        .LOC    WIN_STAT+?WIN_STATUS.VT_SZ.Y    ; Height of virtual
        .DWORD  0                               ;   terminal (chars).
        .LOC    WIN_STAT+?WIN_STATUS.VT_TYPE    ; Virtual terminal type.
        .DWORD  0
        .LOC    WIN_STAT+?WIN_STATUS.PIXEL_DEPTH  ; Bits per pixel in
        .WORD   0                                 ;   pixel map.
        .LOC    WIN_STAT+?WIN_STATUS.INPUT_BUF_SZ ; Length of input
        .WORD   0                                 ;   buffer (bytes).
        .LOC    WIN_STAT+?WIN_STATUS.PAL_TYPE   ; Palette type.
        .DWORD  0
        .LOC    WIN_STAT+?WIN_STATUS_LEN_REV1   ; End of subpacket.
```

Figure A–20. Listing of Program CREATE_WINDOW.SR (continued)

```
        ; Define some masks for the ?WIN_DEFINE_PORTS flag word.

        SCAN_SZ_X_MSK =  1S?WIN_DPORT.FLAGS.SCAN_SZ_X
        SCAN_SZ_Y_MSK =  1S?WIN_DPORT.FLAGS.SCAN_SZ_Y
        VIEW_ORG_X_MSK = 1S?WIN_DPORT.FLAGS.VIEW_ORG_X
        VIEW_ORG_Y_MSK = 1S?WIN_DPORT.FLAGS.VIEW_ORG_Y

        ; Set up the ?WIN_DEFINE_PORTS subpacket.

WIN_DPORT: .BLK  ?WIN_DPORT_LEN                 ; Allocate enough space for
                                                ;   ?WIN_DEFINE_PORTS
                                                ;   subpacket.
        .LOC    WIN_DPORT+?WIN_DPORT.PKT_ID     ; Packet ID.
        .DWORD  ?WIN_DPORT_PKTID
        .LOC    WIN_DPORT+?WIN_DPORT.FLAGS      ; Flags: this call specifies
                                                ;   the dimensions of the
                                                ;   scan port and the
                                                ;   X-coordinate of the view
                                                ;   port's origin.
        .DWORD  SCAN_SZ_X_MSK!SCAN_SZ_Y_MSK!VIEW_ORG_X_MSK
        .LOC    WIN_DPORT+?WIN_DPORT.VIEW_ORG.X ; X-coordinate of view port
        .DWORD  100.                            ;   origin (pixels): 100.
        .LOC    WIN_DPORT+?WIN_DPORT.VIEW_ORG.Y ; Y-coordinate of view port
        .DWORD  0                               ;   origin (pixels):
                                                ;   not set.
        .LOC    WIN_DPORT+?WIN_DPORT.SCAN_ORG.X ; X-coordinate of scan port
        .DWORD  0                               ;   origin (chars):
                                                ;   not set.
        .LOC    WIN_DPORT+?WIN_DPORT.SCAN_ORG.Y ; Y-coordinate of scan port
        .DWORD  0                               ;   origin (chars):
                                                ;   not set.
        .LOC    WIN_DPORT+?WIN_DPORT.SCAN_SZ.X  ; Width of scan port
        .DWORD  80.                             ;   (characters):
                                                ;   80. visible columns.
        .LOC    WIN_DPORT+?WIN_DPORT.SCAN_SZ.Y  ; Height of scan port
                                                ;   (characters):
        .DWORD  24.                             ;   24. visible rows.
        .LOC    WIN_DPORT+?WIN_DPORT_LEN        ; End of subpacket.
```

Figure A–20. Listing of Program CREATE_WINDOW.SR (continued)

```
        ; Set up the ?PROC packet
PROGRAM: .TXT   ":CLI.PR"                   ; Program name of CLI to ?PROC
                                            ;   in the new window.
PROC_PKT: .BLK  ?PLTH                       ; Allocate enough space
                                            ;   for ?PROC packet.
         .LOC    PROC_PKT+?PFLG             ; Process creation options:
         .WORD   0+?PFEX                    ;   Block caller while son
                                            ;   executes.
         .LOC    PROC_PKT+?PPRI             ; Priority of son process is
         .WORD   -1                         ;   same as caller's priority.
         .LOC    PROC_PKT+?PSNM             ; Byte pointer to pathname of
         .DWORD  PROGRAM*2                  ;   program to execute:
                                            ;   ":CLI.PR"
         .LOC    PROC_PKT+?PIPC             ; Address of IPC message
         .DWORD  -1                         ;   header: There is no
                                            ;   initial IPC message.
         .LOC    PROC_PKT+?PNM              ; Simple process name of son
         .DWORD  -1                         ;   is ASCII representation of
                                            ;   PID.
         .LOC    PROC_PKT+?PMEM             ; Son has same logical page
         .DWORD  -1                         ;   maximum as caller.
         .LOC    PROC_PKT+?PDIR             ; Son has same initial working
         .DWORD  -1                         ;   directory as caller.
         .LOC    PROC_PKT+?PCON             ; Byte pointer to son's
         .DWORD  WINDOW_PATH*2              ;   @CONSOLE file: set to
                                            ;   window pathname.
         .LOC    PROC_PKT+?PCAL             ; Son is limited to two
         .WORD   -1                         ;   concurrent system calls.
         .LOC    PROC_PKT+?PWSS             ; Son has no maximum working
         .WORD   -1                         ;   set size.
         .LOC    PROC_PKT+?PUNM             ; Son has same username as
         .DWORD  -1                         ;   caller.
         .LOC    PROC_PKT+?PPRV             ; Son has all caller's file
         .WORD   -1                         ;   access privileges.
         .LOC    PROC_PKT+?PPCR             ; Son can create same maximum
         .WORD   -1                         ;   number of sons as caller.
         .LOC    PROC_PKT+?PWMI             ; Son has no minimum working
         .WORD   -1                         ;   set size.
         .LOC    PROC_PKT+?PIFP             ; Byte pointer to son's @INPUT
         .DWORD  WINDOW_PATH*2              ;   file: set to window
                                            ;   pathname.
         .LOC    PROC_PKT+?POFP             ; Byte pointer to son's
         .DWORD  WINDOW_PATH*2              ;   @OUTPUT file: set to
                                            ;   window pathname.
         .LOC    PROC_PKT+?PLFP             ; Son has same @LIST file as
         .DWORD  -1                         ;   caller.
         .LOC    PROC_PKT+?PDFP             ; Son has same @DATA file as
         .DWORD  -1                         ;   caller.
         .LOC    PROC_PKT+?SMCH             ; Son receives remainder of
         .DWORD  -1                         ;   father's CPU time limit.
         .LOC   PROC_PKT+?PLTH              ; End of packet.
         .END   CREATE_WINDOW
```

Figure A-20. Listing of Program CREATE_WINDOW.SR (concluded)

You must assemble CREATE_WINDOW.SR with a MASM–created symbol table file (.PS file) that includes long names for the offsets in the parameter packets of system call ?WINDOW. Each release of AOS/VS software includes such a file; its default pathname is :UTIL:MASM_32CHAR.PS.

Also, you must execute CREATE_WINDOW.PR on a graphics terminal that is part of the DS/7xxx family of computers. If you don't have this hardware, CREATE_WINDOW.PR will terminate with an *Illegal function for device* error message.

The assembly command is

```
) XEQ MASM/PS=MASM_32CHAR.PS CREATE_WINDOW ⤶
```

The Link command is

```
) XEQ LINK CREATE_WINDOW ⤶
```

The command to execute the program is

```
) XEQ CREATE_WINDOW ⤶
```

When the program begins execution you will see a rectangular window with the word "CLI" in a gray bar at the top of the window. Directly under the gray bar are a familiar log–on message and CLI prompt similar to the following.

AOS/VS CLI Rev 07.56.00.00 19–JAN–88 11:13:18
)

After you finish your CLI session, give the familiar BYE command. This terminates the son CLI process and its father process resumes.

You might consider making a change to program CREATE_WINDOW by turning its window's permanence on just before the user sees it and by turning the window's permanence off just before the program deletes it. However, making this change would require the user to have System Manager privilege turned on at runtime.

Figure A–21 contains the code needed to turn the window's permanence on. This code belongs, in Figure A–20, between the lines

```
        ?WINDOW                                 ; Define view port and
                                                ;   scan port.
        WBR     ERROR                           ; Quit.
```

and

```
        ; Make the window visible to the user using the ?WINDOW system
        ;   call, function code ?WIN_UNHIDE_WINDOW.
```

Figure A–21 is next.

```
        ; Turn permanence on for the window using the ?WINDOW system call,
        ;   function code ?WIN_PERMANENCE_ON.
PERM_ON:
        XLEF    2,WIN_PKT               ; Load word pointer to main
                                        ;   packet in AC0.
        NLDAI   ?WIN_PERMANENCE_ON,0    ; Specify ?WIN_PERMANENCE_ON
        XNSTA   0,?WIN_PKT.FUNC,2       ;   function code in the
                                        ;   main packet.
        WSUB    0,0                     ; Clear AC0.
        WSUB    1,1                     ; Clear AC1.
        XWSTA   0,?WIN_PKT.SUBPKT,2     ; No subpacket for this
                                        ;   function.
        ?WINDOW                         ; Turn permanence on.
        WBR     ERROR                   ; Quit.
```

Figure A–21. Code to Turn Permanence On in Program CREATE_WINDOW

Figure A–22 contains the code needed to turn the window's permanence off. This code belongs, in Figure A–20, between the lines

```
        ; The program pends here until the user terminates the CLI.
        ;   Unless there is an error, the program then deletes the
        ;   window.
```

and

```
        ; Delete the window using the ?WINDOW system call, function code
        ;   ?WIN_DELETE_WINDOW.
```

Figure A–22 is next.

```
        ; Turn permanence off for the window using the ?WINDOW
        ;   call, function code ?WIN_PERMANENCE_OFF.
PERM_OFF:
        XLEF    2,WIN_PKT               ; Load word pointer to the
                                        ;   main packet in AC0.
        NLDAI   ?WIN_PERMANENCE_OFF,0   ; Specify
                                        ;   ?WIN_PERMANENCE_OFF
        XNSTA   0,?WIN_PKT.FUNC,2       ;   function code in the
                                        ;   main packet.
        WSUB    0,0                     ; Clear AC0.
        WSUB    1,1                     ; Clear AC1.
        XWSTA   0,?WIN_PKT.SUBPKT,2     ; No subpacket for this
                                        ;   function.
        ?WINDOW                         ; Turn permanence off.
        WBR     ERROR                   ; Quit.
```

Figure A–22. Code to Turn Permanence Off in Program CREATE_WINDOW

Program Set 12 — GRAPHICS_SAMPLE.SR

Program GRAPHICS_SAMPLE creates a graphics window. Then it receives input from a pointer device and uses it to draw lines and bars. The program issues system calls ?GRAPHICS, ?OPEN, ?PTRDEVICE, ?READ, ?RETURN, and ?WINDOW.

Figure A–23 contains GRAPHICS_SAMPLE.SR.

```
        .TITLE  GRAPHICS_SAMPLE
        .ENT    CREATE_GRAPHICS
        .NREL   0

; CREATE_GRAPHICS -- Creates a graphics window, and calls internal
;                    subroutine DRAW.

; The program assumes that its @CONSOLE file is a window on a pixel map.

CREATE_GRAPHICS:
;  Create a graphics window.

        XLEF    2,WIN_PKT                       ; Load address of main packet.
        WLDAI   ?WIN_PKT_PKTID,0                ; Load WIN_PKT packet ID, and
        XWSTA   0,?WIN_PKT.PKT_ID,2             ;  store in main window packet.
        NLDAI   ?WIN_CREATE_WINDOW,0            ; Specify ?WIN_CREATE_WINDOW
        XNSTA   0,?WIN_PKT.FUNC,2               ;  function code in main packet.
        WSUB    0,0                             ; Clear AC0.
        XNSTA   0,?WIN_PKT.CHAN_NUM,2           ; Set channel number to zero.
        WLDAI   1S?WIN_PKT.FLAGS.IN_PATH,0      ; Specify window pathname by
        XWSTA   0,?WIN_PKT.FLAGS,2              ;  setting flag in main packet.
        XLEFB   0,PMAP*2                        ; Store byte pointer to pathname
        XWSTA   0,?WIN_PKT.PATH,2               ;  in main packet.
        NLDAI   PMAP_PATH_LENGTH,0              ; Store length of pathname in
        XNSTA   0,?WIN_PKT.PATH_BUF_LEN,2       ;  main packet.
        WSUB    0,0                             ; Clear AC0.

        XNSTA   0,?WIN_PKT.PATH_LEN,2           ; Set pathname length to zero.
        XWSTA   0,?WIN_PKT.WIND_ID,2            ; Set window ID to zero.
        XLEF    0,WIN_CRE_SUBPKT                ; Store address of create
        XWSTA   0,?WIN_PKT.SUBPKT,2             ;  subpacket into main packet.

        XLEF    2,WIN_CRE_SUBPKT                ; Load address of WIN_CRE
                                                ;  subpacket.
        WLDAI   ?WIN_CRE_PKTID_REV1,0           ; Load WIN_CRE packet ID, and
        XWSTA   0,?WIN_CRE.PKT_ID,2             ;  store in subpacket.
        WLDAI   NEW_GRP!VT_SZ_X!VT_SZ_Y!PIX!BUF_SZ!UHIDE,0
        XWSTA   0,?WIN_CRE.FLAGS,2              ; Flags: Window starts a new
                                                ;  group. Virtual terminal size,
                                                ;  pixel depth and input buffer
                                                ;  size are specified.  Unhide
                                                ;  the window after creation.

        WLDAI   ?WIN_VT_TYPE_PIXMAP,0           ; Specify pixel map as window's
        XWSTA   0,?WIN_CRE.VT_TYPE,2            ;  virtual terminal type.
        XWLDA   0,WIND_WIDTH                    ; Load window's virtual terminal
        XWSTA   0,?WIN_CRE.VT_SZ.X,2            ;  width into subpacket.
        XWLDA   0,WIND_HEIGHT                   ; Load window's virtual terminal
        XWSTA   0,?WIN_CRE.VT_SZ.Y,2            ;  height into subpacket.
        XLEFB   0,WINDOW_TITLE*2                ; Load byte pointer to window
        XWSTA   0,?WIN_CRE.TITLE,2              ;  title into subpacket.
        NLDAI   TITLE_LEN,0                     ; Load size of window title
        XNSTA   0,?WIN_CRE.TITLE_BUF_LEN,2      ;  buffer into subpacket.
        NLDAI   PIXEL_DEPTH,0                   ; Load pixel depth of virtual
        XNSTA   0,?WIN_CRE.PIXEL_DEPTH,2        ;  pixel map into subpacket.
        WLDAI   BUFFER_SZ,0                     ; Load size of window's input
        XNSTA   0,?WIN_CRE.INPUT_BUF_SZ,2       ;  buffer into subpacket.
        NLDAI   NAME_LEN,0                      ; Load size of window name
        XNSTA   0,?WIN_CRE.WNAME_BUF_LEN,2      ;  buffer into subpacket.
        XLEFB   0,WINDOW_NAME*2                 ; Load byte pointer to window
        XWSTA   0,?WIN_CRE.WNAME,2              ;  name buffer into subpacket.
```

Figure A–23. Listing of Program GRAPHICS_SAMPLE.SR (continued)

```
        WLDAI   ?WIN_PALETTE_TYPE_USER,0    ; Specify user definable
        XWSTA   0,?WIN_CRE.PAL_TYPE,2       ;  palette as window's palette
                                            ;  type.
        NLDAI   ?WIN_BORDER_TYPE_TITLE,0    ; Specify title border as
        XNSTA   0,?WIN_CRE.BORDER_TYPE,2    ;  window's border type.
        WSUB    0,0                         ; Clear AC0.
        XNSTA   0,?WIN_CRE.RSV,2            ; Set reserved value to zero.
        XWSTA   0,?WIN_CRE.VIEW_ORG.X,2     ; Clear view port origin
        XWSTA   0,?WIN_CRE.VIEW_ORG.Y,2     ;  coordinate fields.
        WSUB    0,0                         ; AC0 must be set to zero.
        WSUB    1,1                         ; AC1 must be set to zero.
        XLEF    2,WIN_PKT                   ; Load address of WIN packet.
        ?WINDOW                             ; Create a graphics window.
        WBR     CREATE_ERROR                ; Handle an error.
; Use window ID as window specifier for remaining ?WINDOW system calls.
        XWLDA   0,?WIN_PKT.WIND_ID,2        ; Load window ID, and store
        XWSTA   0,WIND_ID                   ;  it in location WIND_ID.
        WLDAI   1S?WIN_PKT.FLAGS.IN_WIND_ID,0 ; Use window ID as specifier
        XWSTA   0,?WIN_PKT.FLAGS,2          ;    by setting flag in the
                                            ;    main packet.
; Get the window's pathname.
        XLEF    2,WIN_PKT                   ; Load word pointer to the
                                            ;  main packet in AC2.
        NLDAI   ?WIN_GET_WINDOW_NAME,0      ; Specify ?WIN_GET_WINDOW_NAME
        XNSTA   0,?WIN_PKT.FUNC,2           ;  function code in main packet.
        WSUB    0,0                         ; Clear subpacket pointer
        XWSTA   0,?WIN_PKT.SUBPKT,2         ;  in the main packet.
        XLEFB   0,WINDOW_PATH*2             ; Pathname buffer.
        XWSTA   0,?WIN_PKT.PATH,2
        NLDAI   256.,0                      ; Set pathname buffer length.
        XNSTA   0,?WIN_PKT.PATH_BUF_LEN,2
        WSUB    0,0                         ; Clear AC0.
        WSUB    1,1                         ; Clear AC1.
        ?WINDOW                             ; Get status.  AOS/VS returns
                                            ;  information in subpacket.
        WBR     CREATE_ERROR                ; Quit.
        WSUB    0,0                         ; Clear AC0.
        XWSTA   0,?WIN_PKT.PATH,2           ; Clear pathname in the main
                                            ;  packet.
        XNSTA   0,?WIN_PKT.PATH_BUF_LEN,2   ; Clear pathname length.
        XNSTA   0,?WIN_PKT.PATH_LEN,2
        WBR     GRAPH_OPEN                  ; Continue setting up the
                                            ;  window.
CREATE_ERROR:
        XJMP    CREATE_ERROR_RTN            ; Error routine handler.
```

Figure A–23. Listing of Program GRAPHICS_SAMPLE.SR (continued)

```
GRAPH_OPEN:
; Open the graphics window for output.
        XLEF    2,GRAPH_PKT                     ; Load address of graphics pkt.
        WLDAI   ?GRAPH_PKTID,0                  ; Load GRAPH_PKT packet ID,
        XWSTA   0,?GRAPH_PKT.PKT_ID,2           ;  and store it in the main
                                                ;  graphics packet.
        NLDAI   ?GRAPH_OPEN_WINDOW_PIXELMAP,0 ; Specify ?GRAPH_OPEN
        XNSTA   0,?GRAPH_PKT.FUNC,2             ;    function code in the
                                                ;    main graphics packet.
        WSUB    0,0                             ; Clear AC0.
        XNSTA   0,?GRAPH_PKT.RSV,2              ; Set reserved value to zero.
        XWSTA   0,?GRAPH_PKT.PIXMAP_ID,2        ; Set pixelmap ID to zero.
        XLEF    0,GRAPH_OPEN_SUBPKT             ; Load address of GRAPH_OPEN
        XWSTA   0,?GRAPH_PKT.SUBPKT,2           ;  subpacket into main packet.
        XLEF    2,GRAPH_OPEN_SUBPKT             ; Load address of GRAPH_OPEN
                                                ;  subpacket.
        WLDAI   ?GRAPH_OPEN_PKTID,0             ; Load GRAPH_OPEN packet ID,
        XWSTA   0,?GRAPH_OPEN.PKT_ID,2          ;  store it in subpacket.
        WSUB    0,0                             ; Clear AC0.
        XNSTA   0,?GRAPH_OPEN.RSV,2             ; Set reserved value to zero.
        XNSTA   0,?GRAPH_OPEN.CHAN_NUM,2        ; Clear channel number.
        WLDAI   1S?GRAPH_OPEN.FLAGS.IN_WIND_ID,0
        XWSTA   0,?GRAPH_OPEN.FLAGS,2
        WSUB    0,0
        XWSTA   0,?GRAPH_OPEN.PATH,2
        XNSTA   0,?GRAPH_OPEN.PATH_BUF_LEN,2
        XNSTA   0,?GRAPH_OPEN.PATH_LEN,2
        XWLDA   0,WIND_ID
        XWSTA   0,?GRAPH_OPEN.WIND_ID,2
        WSUB    0,0                             ; AC0 must be set to zero.
        WSUB    1,1                             ; AC1 must be set to zero.
        XLEF    2,GRAPH_PKT                     ; Load address of graphics pkt.
        ?GRAPHICS                               ; Open window for output.
        WBR     CREATE_ERROR_RTN                ; Handle an error.
        XWLDA   0,?GRAPH_PKT.PIXMAP_ID,2        ; Load GIS form ID, and store
        XWSTA   0,FORM_ID                       ;  it in location FORM_ID.
OPEN:
; Open the graphics window for input.
        XLEF    2,OPEN_PKT
        WSUB    0,0
        XNSTA   0,?ICH,2
        NLDAI   ?OFIN!?ICRF!?RTDY,0
        XNSTA   0,?ISTI,2
        WSUB    0,0
        XNSTA   0,?ISTO,2
        WADC    0,0
        XNSTA   0,?IMRS,2
        XWSTA   0,?IBAD,2
        XNSTA   0,?IRES,2
        XNSTA   0,?IRCL,2
        XWSTA   0,?IDEL,2
        XLEFB   0,WINDOW_PATH*2
        XWSTA   0,?IFNP,2
        WSUB    0,0                             ; AC0 must be set to zero.
        WSUB    1,1                             ; AC1 must be set to zero.
        XLEF    2,OPEN_PKT                      ; Load address of ?OPEN packet.
        ?OPEN                                   ; Open window for input.
        WBR     CREATE_ERROR_RTN                ; Handle an error.
        XNLDA   0,?ICH,2                        ; Load channel number, and
        XNSTA   0,CHANNEL                       ;  store in location CHANNEL.
```

Figure A–23. Listing of Program GRAPHICS_SAMPLE.SR (continued)

```
WRITE_PAL:
; Initialize the palette.
        XLEF    2,GRAPH_PKT
        NLDAI   ?GRAPH_WRITE_PALETTE,0
        XNSTA   0,?GRAPH_PKT.FUNC,2
        XLEF    0,WRPAL_SUBPKT
        XWSTA   0,?GRAPH_PKT.SUBPKT,2

        XLEF    2,WRPAL_SUBPKT
        WLDAI   ?GRAPH_WRPAL_PKTID,0
        XWSTA   0,?GRAPH_WRPAL.PKT_ID,2
        WLDAI   ?GRAPH_WRPAL_ARRAY_PKTID,0
        XWSTA   0,?GRAPH_WRPAL.ARRAY_PKT_ID,2
        WLDAI   0,0
        XWSTA   0,?GRAPH_WRPAL.FIRST_PAL_INDEX,2
        WLDAI   4,0
        XWSTA   0,?GRAPH_WRPAL.ENTRY_CNT,2
        XLEF    0,PAL_ARRAY
        XWSTA   0,?GRAPH_WRPAL.PH0_ARRAY,2
        XWSTA   0,?GRAPH_WRPAL.PH1_ARRAY,2

        WSUB    0,0                     ; AC0 must be set to zero.
        WSUB    1,1                     ; AC1 must be set to zero.
        XLEF    2,GRAPH_PKT             ; Load address of the graphics
                                        ;  packet.
        ?GRAPHICS                       ; Set up the palette.
        WBR     CREATE_ERROR_RTN        ; Handle an error.
; Call internal subroutine DRAW.
        XJSR    DRAW                    ; Call subroutine DRAW.
        WBR     CREATE_ERROR_RTN        ; Handle an error.

        WSUB    2,2                     ; Set flags for normal return.
        WBR     BYE                     ; Branch to good return.
CREATE_ERROR_RTN:
        WLDAI  ?RFEC!?RFCF!?RFER,2      ; Error flags: Error code is in
                                        ;  AC0 (?RFEC), message is in
                                        ;  CLI format (?RFCF), caller
                                        ;  should handle it as an error
                                        ;  (?RFER).
BYE:    ?RETURN                         ; Return to the calling process.
        WBR CREATE_ERROR_RTN
```

Figure A–23. Listing of Program GRAPHICS_SAMPLE.SR (continued)

```
; DRAW -- Internal subroutine that gets pointer events from the user,
;         and draws lines and bars.
; Button 1 starts and ends drawing lines.  Button 2 starts and ends
;  drawing bars.  Double click on button 2 exits the program.
;
; Set up storage for program variables.
        LINE_OP =       1                       ; Line drawing flag.
        BAR_OP =        2                       ; Bar drawing flag.
        DRAW_OP:        0                       ; Drawing operation,
                                                ;  initialized to no-operation.
        BAR_ANCHOR_X:   0                       ; Bar anchor point coordinates.
        BAR_ANCHOR_Y:   0
        OP_MSK:         3                       ; GIS II Operation Mask: bits 1
                                                ;  and 2 of pixel are affected.
        COMBO_RULE:     6                       ; GIS II Combination Rule:
                                                ;  destination := source XOR
                                                ;                 destination.
        LINE_CTRL:      0                       ; GIS II Line Control: draw all
                                                ;  pixels, and both endpoints.
        LINE_STYLE:     -1                      ; GIS II Line Style: solid
                                                ;  line - draw all pixels in
                                                ;  the foreground.
        BACKGROUND:     0                       ; Background color is palette
                                                ;  entry zero.
        FOREGROUND:     1                       ; Foreground color is palette
                                                ;  entry one.
DRAW:
        WSSVR   0                               ; Save return block on stack.
        LDAFP   3                               ; Load frame pointer into AC3.
```

Figure A–23. Listing of Program GRAPHICS_SAMPLE.SR (continued)

```
; Set pointer events - receive movement events, and button down and
;  double-click events from buttons 1 and 2.
        XLEF    2,PTRDEV_PKT                    ; Load pointer to PTRDEV_PKT.
        WLDAI   ?PTRDEV_PKTID,0                 ; Load PTRDEV_PKT packet ID, and
        XWSTA   0,?PTRDEV_PKT.PKT_ID,2          ;  store it in main packet.

        NLDAI   ?PTRDEV_SET_EVENTS,0            ; Load SET_EVENTS function code,
        XNSTA   0,?PTRDEV_PKT.FUNC,2            ;  and store it in main packet.

        WLDAI   1S?PTRDEV_PKT.FLAGS.IN_WIND_ID,0  ; Use window ID as window
        XWSTA   0,?PTRDEV_PKT.FLAGS,2             ;  specifier.

        WSUB    0,0                             ; Clear AC0.

        XWSTA   0,?PTRDEV_PKT.PATH,2             ; Set window path to zero.
        XNSTA   0,?PTRDEV_PKT.PATH_BUF_LEN,2     ; Set window path length to 0.
        XNSTA   0,?PTRDEV_PKT.PATH_LEN,2         ; Set return path length to 0.

        XWLDA   0,WIND_ID                       ; Load window ID, and store
        XWSTA   0,?PTRDEV_PKT.WIND_ID,2         ;  store it in main packet.
        XLEF    0,PTRDEV_EVTS_PKT               ; Load pointer to EVENTS sub-
        XWSTA   0,?PTRDEV_PKT.SUBPKT,2          ;  packet, and store it in
                                                ;  the main packet.
        XLEF    2,PTRDEV_EVTS_PKT               ; Load pointer to EVENTS
                                                ;  subpacket.
        WLDAI   ?PTRDEV_SET_EVTS_PKTID,0        ; Load SET_EVENTS subpacket
        XWSTA   0,?PTRDEV_SET_EVTS.PKT_ID,2     ;  ID, and store in subpacket.

        WLDAI   PTR_MOV!PTR_BTN_DWN!PTR_DBL,0   ; Ask for movement, button
        XWSTA   0,?PTRDEV_SET_EVTS.EVTS,2       ;  down, and double-click
                                                ;  events.

        WLDAI   PTR_BTN1!PTR_BTN2,0             ; Ask for buttons 1 and 2.
        XWSTA   0,?PTRDEV_SET_EVTS.BTNS,2

        WSUB    0,0                             ; AC0 must be set to zero.
        WSUB    1,1                             ; AC1 must be set to zero.
        XLEF    2,PTRDEV_PKT                    ; Load pointer to PTRDEV packet.

        ?PTRDEVICE                              ; Set pointer events.
        WBR     DRAW_ERROR                      ; Handle an error.
```

Figure A–23. Listing of Program GRAPHICS_SAMPLE.SR (continued)

```
; Write GIS II Form attributes.
        WLDAI   0,0                     ; Set operation mask.
        XWLDA   1,FORM_ID               ; Load GIS II form ID.
        XLEF    2,OP_MSK                ; Load address of operation
                                        ;  mask value.
        WGWRATTR  GISERROR              ; GIS II write attributes.

        WLDAI   1,0                     ; Set combination rule.
        XWLDA   1,FORM_ID               ; Load GIS II form ID.
        XLEF    2,COMBO_RULE            ; Load address of combination
                                        ;  rule value.
        WGWRATTR  GISERROR              ; GIS II write attributes.

        WLDAI   2,0                     ; Set line control word.
        XWLDA   1,FORM_ID               ; Load GIS II form ID.
        XLEF    2,LINE_CTRL             ; Load address of line control
                                        ;  value.
        WGWRATTR  GISERROR              ; GIS II write attributes.

        WLDAI   3,0                     ; Set line foreground color.
        XWLDA   1,FORM_ID               ; Load GIS II form ID.
        XLEF    2,FOREGROUND            ; Load address of foreground
                                        ;  color value.
        WGWRATTR  GISERROR              ; GIS II write attributes.

        WLDAI   4,0                     ; Set line background color.
        XWLDA   1,FORM_ID               ; Load GIS II form ID.
        XLEF    2,BACKGROUND            ; Load address of background
                                        ;  color value.
        WGWRATTR  GISERROR              ; GIS II write attributes.

        WLDAI   5,0                     ; Set line style.
        XWLDA   1,FORM_ID               ; Load GIS II form ID.
        XLEF    2,LINE_STYLE            ; Load address of line style
                                        ;  value.
        WGWRATTR  GISERROR              ; GIS II write attributes.
; RFLOOD window with background color.
        WSUB    0,0                     ; Clear AC0.
        XWSTA   0,RFLOOD_PKT+ULC_X      ; Set ULC of RFLOOD rectangle
        XWSTA   0,RFLOOD_PKT+ULC_Y      ;  to window coordinates (0,0).
        XWLDA   0,WIND_WIDTH            ; Set width of RFLOOD rectangle
        XWSTA   0,RFLOOD_PKT+EXTENT_X   ;  to the width of the window.
        XWLDA   0,WIND_HEIGHT           ; Set height of RFLOOD rectangle
        XWSTA   0,RFLOOD_PKT+EXTENT_Y   ;  to the height of the window.

        XWLDA   0,BACKGROUND            ; Load background color.
        XWLDA   1,FORM_ID               ; Load GIS II form ID.
        XLEF    2,RFLOOD_PKT            ; Load RFLOOD packet.
        WGRFLOOD  GISERROR              ; RFLOOD window.
```

Figure A–23. Listing of Program GRAPHICS_SAMPLE.SR (continued)

```
PROCESS_EVENT:
; Get events from the window input buffer, and branch to process
;  button down, movement and double-click events.  (Keystroke events
;  and all other pointer events are ignored.)
        WSUB    0,0                     ; Clear AC0.
        XWLDA   1,CHANNEL               ; Load input channel into AC1.
        XLEF    2,EVENT_PKT             ; Load pointer to event packet.
        XJSR    GET_NEXT_EVENT          ; Get next event in the input
                                        ;  buffer.
        WBR     DRAW_ERROR              ; Handle an error.

        XWLDA   0,EVENT.TYPE,2          ; Load event type.
        WSNEI   61,0                    ; Was event a button down?
        WBR     BUTTON_DWN              ; Yes, handle event.

        WSNEI   60,0                    ; Was event a movement event?
        WBR     MOVE                    ; Yes, handle event.

        WSNEI   67,0                    ; Was event a double click?
        WBR     DBL_CLICK               ; Yes, handle event.

        WBR     PROCESS_EVENT           ; Other event types are
                                        ;  ignored -- continue
                                        ;  processing the next event.
DRAW_ERROR:
        XJMP    DRAW_ERROR_RTN          ; Handle an error.
BUTTON_DWN:
;  Process button events.
        XWLDA   0,EVENT.BTN,2           ; Load button number.
        WSNEI   61,0                    ; Was it button 1?
        WBR     BTN1_DWN                ; Yes, process button 1
                                        ;  down event.
        WSNEI   62,0                    ; Was it button 2?
        WBR     BTN2_DWN                ; Yes, process button 2
                                        ;  down event.
        WBR     PROCESS_EVENT           ; Other buttons are ignored --
                                        ;  continue processing the
                                        ;  next event.
BTN1_DWN:
; Process button 1 event -- if no drawing operation is currently
;  in progress, then set the line anchor point to start drawing a line.
;  If currently drawing a line, then end drawing line.  If currently
;  drawing a bar, then the event is ignored.
        XWLDA   0,DRAW_OP               ; Load current drawing
                                        ;  operation.
        WSNEI   0,0                     ; No operation in progress?
        WBR     SET_LINE_ANCHOR         ; Set anchor for line drawing.
        WSEQI   LINE_OP,0               ; Currently drawing a line?
        WBR     PROCESS_EVENT           ; No, continue.
        WSUB    0,0                     ; Yes, reset draw operation to
        XWSTA   0,DRAW_OP               ;  end drawing current line.
        WBR     PROCESS_EVENT           ; Continue processing the
                                        ;  next event.
```

Figure A–23. Listing of Program GRAPHICS_SAMPLE.SR (continued)

```
BTN2_DWN:
; Process button 2 event -- if no drawing operation is currently
;  in progress, then set the bar anchor point to start drawing a bar.
;  If currently drawing a bar, then end drawing bar.  If currently
;  drawing a line, then the event is ignored.
        XWLDA   0,DRAW_OP                   ; Load current drawing
                                            ;  operation.
        WSNEI   0,0                         ; No operation in progress?
        WBR     SET_BAR_ANCHOR              ; Set anchor for bar drawing.
        WSEQI   BAR_OP,0                    ; Currently drawing a bar?
        WBR     PROCESS_EVENT               ; No, continue.
        WSUB    0,0                         ; Yes, reset draw operation to
        XWSTA   0,DRAW_OP                   ;  end drawing bar.
        WBR     PROCESS_EVENT               ; Continue processing the
                                            ;  next event.
MOVE:
; Process movement event -- redraw line or bar depending on the current
;  drawing operation.
        XWLDA   0,DRAW_OP                   ; Load current drawing
                                            ;  operation.
        WSNEI   LINE_OP,0                   ; Currently drawing a line?
        WBR     REDRAW_LINE                 ; Yes, redraw line.
        WSNEI   BAR_OP,0                    ; Currently drawing a bar?
        WBR     REDRAW_BAR                  ; Yes, redraw bar.
        WBR     PROCESS_EVENT               ; Continue processing the
                                            ;  next event.
DBL_CLICK:
; Process double click event -- double click on button 2 ends program.
        XWLDA   0,EVENT.BTN,2               ; Load button.
        WSEQI   62,0                        ; Was it button 2?
        WBR     PROCESS_EVENT               ; No, continue.
        XJMP    DRAW_GOOD_RTN               ; Yes, exit.
SET_LINE_ANCHOR:
; Set the line anchor point, i.e., set the initial endpoint of the line.
;  The final endpoint is initially set to the same location, but will be
;  updated with each pointer movement event.
        WLDAI   LINE_OP,0                   ; Set current drawing operation
        XWSTA   0,DRAW_OP                   ;  to line drawing.
        XWLDA   0,EVENT.LOC_X,2             ; Load x-coordinate of pointer.
        XWSTA   0,LINE_PKT+X1               ; Set anchor point x-coordinate.
        XWSTA   0,LINE_PKT+X2               ; Set endpoints the same.
        XWLDA   0,EVENT.LOC_Y,2             ; Load y-coordinate of pointer.
        XWSTA   0,LINE_PKT+Y1               ; Set anchor point y-coordinate.
        XWSTA   0,LINE_PKT+Y2               ; Set endpoints the same.
        WBR     PROCESS_EVENT               ; Continue processing the next
                                            ;  event.
```

Figure A–23. Listing of Program GRAPHICS_SAMPLE.SR (continued)

```
SET_BAR_ANCHOR:
; Set bar anchor point -- the bar anchor point is a fixed corner of the
;  bar.  The ULC (upper-left corner) of the bar is initially set to this
;  anchor point.  The anchor point may correspond to any other corner of
;  the bar depending on the position of the pointer.
        WLDAI   BAR_OP,0                ; Set current drawing operation
        XWSTA   0,DRAW_OP               ;  to bar drawing.

        XWLDA   0,EVENT.LOC_X,2         ; Load x-coordinate of pointer.
        XWSTA   0,RFLOOD_PKT+ULC_X      ; Set ULC x-coordinate.
        XWSTA   0,BAR_ANCHOR_X          ; Set anchor point x-coordinate.
        XWLDA   0,EVENT.LOC_Y,2         ; Load y-coordinate of pointer.
        XWSTA   0,RFLOOD_PKT+ULC_Y      ; Set ULC y-coordinate.
        XWSTA   0,BAR_ANCHOR_Y          ; Set anchor point y-coordinate.

        WSUB    0,0                     ; Clear AC0.
        XWSTA   0,RFLOOD_PKT+EXTENT_X   ; Set RFLOOD rectangle extents
        XWSTA   0,RFLOOD_PKT+EXTENT_Y   ;  to zero.
        WBR     PROCESS_EVENT           ; Continue processing the next
                                        ;  event.
REDRAW_LINE:
; Redraw the line -- undraw the old line, update the location of the
;  final endpoint with the location of the cursor, and draw the new
;  line.  It is assumed that the combination rule set in the GIS II form
;  is the XOR rule.  The line is undrawn by simply drawing it again.
        WLDAI   1,0                     ; Draw one line segment.
        XWLDA   1,FORM_ID               ; Load GIS II form ID.
        XLEF    2,LINE_PKT              ; Load pointer to poly line
                                        ;  packet.
        WGPLINE GISERROR                ; GIS II poly line.

        XLEF    2,EVENT_PKT             ; Load event packet.
        XWLDA   0,EVENT.LOC_X,2         ; Load pointer x-coordinate.
        XWSTA   0,LINE_PKT+X2           ; Set new endpoint x-coordinate.
        XWLDA   0,EVENT.LOC_Y,2         ; Load pointer y-coordinate.
        XWSTA   0,LINE_PKT+Y2           ; Set new endpoint y-coordinate.

        WLDAI   1,0                     ; Draw one line segment.
        XWLDA   1,FORM_ID               ; Load GIS II form ID.
        XLEF    2,LINE_PKT              ; Load pointer to poly line
                                        ;  packet.
        WGPLINE GISERROR                ; GIS II poly line.

        XJMP    PROCESS_EVENT           ; Continue processing the next
                                        ;  event.
```

Figure A-23. Listing of Program GRAPHICS_SAMPLE.SR (continued)

```
REDRAW_BAR:
; Redraw the bar -- undraw the bar, update the ULC and extents using
;  the pointer location, and draw the new bar.  The new ULC will be
;
;  MIN( anchor coordinates, pointer coordinates ).
;
;  The new extents will be
;
;  ABS( pointer coordinates - anchor coordinates ).
;
;  It is assumed that the combination rule set in the GIS II form is the
;  XOR rule.  The bar is undrawn by simply drawing it again.
        XWLDA   0,FOREGROUND                ; Load foreground color.
        XWLDA   1,FORM_ID                   ; Load GIS II form ID.
        XLEF    2,RFLOOD_PKT                ; Load pointer to RFLOOD packet.
        WGRFLOOD  GISERROR                  ; GIS II RFLOOD

        XLEF    2,EVENT_PKT                 ; Load pointer to event packet.
        XWLDA   0,EVENT.LOC_X,2             ; Load x-coordinate of pointer.
        XWSUB   0,BAR_ANCHOR_X              ; Subtract from anchor x-coord.
        WSGE    0,0                         ; Is value greater than zero?
        WNEG    0,0                         ; No, take absolute value.
        XWSTA   0,RFLOOD_PKT+EXTENT_X       ; Store as width of new
                                            ;  rectangle.

        XWLDA   0,EVENT.LOC_Y,2             ; Load y-coordinate of pointer.
        XWSUB   0,BAR_ANCHOR_Y              ; Subtract from anchor y-coord.
        WSGE    0,0                         ; Is value greater than zero?
        WNEG    0,0                         ; No, take absolute value.
        XWSTA   0,RFLOOD_PKT+EXTENT_Y       ; Store as height of new
                                            ;  rectangle.

        XWLDA   0,EVENT.LOC_X,2             ; Load x-coordinate of pointer.
        XWLDA   1,BAR_ANCHOR_X              ; Load x-coordinate of anchor.
        WSGE    0,1                         ; Is pointer greater than
                                            ;  anchor?
        WMOV    0,1                         ; No, move pointer value into
                                            ;  AC1.
        XWSTA   1,RFLOOD_PKT+ULC_X          ; Store minimum value as
                                            ;  RFLOOD ULC.

        XWLDA   0,EVENT.LOC_Y,2             ; Load y-coordinate of pointer.
        XWLDA   1,BAR_ANCHOR_Y              ; Load y-coordinate of anchor.
        WSGE    0,1                         ; Is pointer greater than
                                            ;  anchor?
        WMOV    0,1                         ; No, move pointer value into
                                            ;  AC1.
        XWSTA   1,RFLOOD_PKT+ULC_Y          ; Store minimum value as
                                            ;  RFLOOD ULC.

        XWLDA   0,FOREGROUND                ; Load foreground color.
        XWLDA   1,FORM_ID                   ; Load GIS II form ID.
        XLEF    2,RFLOOD_PKT                ; Load pointer to RFLOOD packet.
        WGRFLOOD  GISERROR                  ; GIS II RFLOOD

        XJMP    PROCESS_EVENT               ; Continue processing the next
                                            ;  event.

GISERROR:
DRAW_ERROR_RTN:
        XWLDA   0,?OAC0,3                   ; Load error into caller's AC0.
        WBR     DRAW_RTN                    ; Return.

DRAW_GOOD_RTN:
        XWADI   1,?ORTN,3                   ; Increment PC for good return.

DRAW_RTN:
        WRTN                                ; Return to caller.
```

Figure A–23. Listing of Program GRAPHICS_SAMPLE.SR (continued)

```
; GET_NEXT_EVENT -- Returns next event from window input buffer.

; On input, AC2 contains a pointer to an event packet.  On output, the
;  packet contains information about the event.

; Temporary stack variables:

        CNT =         2                          ; Keystroke count.
        READ_PKT =    CNT+2                      ; ?READ packet.
        SCREEN_PKT =  READ_PKT+?IBLT             ; Screen edit subpacket.
        STK_SZ =      (SCREEN_PKT+?ESCR+3)/2     ; Temporary variable stack
                                                 ;  space.

GET_NEXT_EVENT:

        WSSVR   STK_SZ                           ; Save return block on stack.
        LDAFP   3                                ; Load frame pointer into AC3.

; If buffer is empty, issue ?READ to read in all events in the window
;  input buffer.  Otherwise, scan buffer for next event.

        XWLDA   0,CURR_EVENT                     ; Load current event byte
                                                 ;  pointer.
        XWLDA   1,EOB                            ; Load EOB byte pointer.
        WSGE    0,1                              ; End of buffer?
        WBR     SCAN_BUFFER                      ; No, scan buffer for the
                                                 ;  next event.

; Buffer is empty.  Set up a greedy ?READ to get all events in the
;  window input buffer.

        XLEF    2,READ_PKT,3                     ; Load pointer to ?READ packet.
        XNLDA   0,CHANNEL                        ; Load input channel, and
        XNSTA   0,?ICH,2                         ;  store it in ?READ packet.
        NLDAI   ?IPKL!?IBIB,0                    ; Status flags: extended packet
        XNSTA   0,?ISTI,2                        ;  (?IPKL), and binary read
                                                 ;  (?IBIB).
        XLEFB   0,BUFFER*2                       ; Load byte pointer to buffer,
        XWSTA   0,?IBAD,2                        ;  and store it in ?READ packet.
        NLDAI   BUFFER_SZ,0                      ; Specify buffer size as record
        XNSTA   0,?IRCL,2                        ;  length in ?READ packet.
        WSUB    0,0                              ; Clear AC0.
        XNSTA   0,?IRLR,2                        ; Set return record length to
                                                 ;  zero.
        XNSTA   0,?IRNW,2                        ; Reserved value set to zero.
        XWSTA   0,?IRNH,2                        ; Set record number to zero.
        XLEF    0,SCREEN_PKT,3                   ; Load pointer to screen
                                                 ;  subpacket.
        WIORI   1S0,0                            ; Set high bit to one.
        XWSTA   0,?ETSP,2                        ; Store in ?READ packet.

        XLEF    2,SCREEN_PKT,3                   ; Load pointer to screen
                                                 ;  subpacket.
        NLDAI   ?ESGT!?ESNE,0                    ; Specify GREEDY READ (?ESGT)
        XNSTA   0,?ESFC,2                        ;  in the screen subpacket
                                                 ;  flags word.
        WSUB    0,0                              ; Clear AC0.
        XNSTA   0,?ESEP,2                        ; Set relative and initial
        XNSTA   0,?ESCR,2                        ;  cursor positions to zero.

        WSUB    0,0                              ; Clear AC0.
        WSUB    1,1                              ; Clear AC1.
        XLEF    2,READ_PKT,3                     ; Load pointer to READ packet.
        ?READ                                    ; GREEDY READ.
        WBR     GET_ERROR                        ; Handle an error.
```

Figure A–23. Listing of Program GRAPHICS_SAMPLE.SR (continued)

```
; ?READ pends until at least one event has been read.  Reset buffer
;  pointers, and begin scanning the buffer.

        XLEFB   0,BUFFER*2                      ; Load byte pointer to buffer.
        XWSTA   0,CURR_EVENT                    ; Set current event to buffer
                                                ;  start.
        XNLDA   1,?IRLR,2                       ; Load returned record length.
        WADD    0,1                             ; Calculate new end-of-buffer.
        XWSTA   1,EOB                           ; Reset end-of-buffer.
        WBR     SCAN_BUFFER                     ; Scan buffer for next event.

GET_ERROR:
        XJMP    GET_ERROR_RTN                   ; Handle an error.

SCAN_BUFFER:

; Scan the buffer.  AC1 contains a byte pointer to the current event, and
;  AC2 contains a byte pointer to the end of the buffer.

        WSUB    0,0                             ; Clear AC0.
        XWSTA   0,CNT,3                         ; Set keystroke count to zero.
        XWLDA   0,CURR_EVENT                    ; Load pointer to current
                                                ;  event.
        XWLDA   1,EOB                           ; Load end-of-buffer pointer.

SCAN:
; Scan the buffer for pointer event header codes.

        WLDB    0,2                             ; Load current byte.
        WSEQI   36,2                            ; Is it <036> ?
        WBR     KEYS_EVENT                      ; No, process keystroke event.
        WINC    0,0                             ; Advance byte pointer.
        WSLT    0,1                             ; End of buffer?
        WBR     RTN_KEYS_EVENT                  ; Yes, return keystroke event.
        WLDB    0,2                             ; Load next byte.
        WSNEI   157,2                           ; Is it <157> ?
        WBR     PTR_EVENT                       ; Yes, process pointer event.
        XWADI   1,CNT,3                         ; Increment keystroke count.

KEYS_EVENT:
; Keystroke events -- count keystrokes and continue scanning.

        XWADI   1,CNT,3                         ; Increment keystroke count.
        WINC    0,0                             ; Advance byte pointer.
        WSLT    0,1                             ; End of buffer?
        WBR     RTN_KEYS_EVENT                  ; Yes, return keystroke event.
        WBR     SCAN                            ; No, continue scanning buffer.

PTR_EVENT:
; Pointer events -- if event delimites a keystroke event, return
;  keystrokes.  Otherwise, process pointer event.

        XWLDA   0,CNT,3                         ; Load keystroke count.
        WSEQ    0,0                             ; Any keystrokes in buffer?
        WBR     RTN_KEYS_EVENT                  ; Yes, return keystroke event.

        XWLDA   0,CURR_EVENT                    ; Load pointer to current
                                                ;  event.
        WINC    0,0                             ; Advance byte pointer.
        WINC    0,0                             ; Advance byte pointer.
        WLDB    0,2                             ; Load next byte.
        WSNEI   60,2                            ; Is current event a movement
                                                ;  event?
        WBR     MOVE_EVENT                      ; Yes, process movement event.
        WBR     RTN_PTR_EVENT                   ; No, return pointer event now.
```

Figure A–23. Listing of Program GRAPHICS_SAMPLE.SR (continued)

```
MOVE_EVENT:
; Compress consecutive movement events -- advance current event pointer
;  to point to the last movement event.
        XWLDA   0,CURR_EVENT                    ; Load current event pointer.
        WADDI   32.,0                           ; Advance pointer to end of
                                                ;  next pointer event.
        WSLT    0,1                             ; Space for another pointer
                                                ;  event?
        WBR     RTN_PTR_EVENT                   ; No, return this event.
        XWLDA   0,CURR_EVENT                    ; Load current event pointer.
        WADDI   16.,0                           ; Advance pointer to end of
                                                ;  event.
        WLDB    0,2                             ; Load byte.
        WSEQI   36,2                            ; Is it <036> ?
        WBR     RTN_PTR_EVENT                   ; No, return current pointer
                                                ;  event.
        WINC    0,0                             ; Advance byte pointer.
        WLDB    0,2                             ; Load next byte.
        WSEQI   157,2                           ; Is it <157> ?
        WBR     RTN_PTR_EVENT                   ; No, return current pointer
                                                ;  event.
        WINC    0,0                             ; Advance byte pointer.
        WLDB    0,2                             ; Load next byte.
        WSEQI   60,2                            ; Is next event a movement
                                                ;  event?
        WBR     RTN_PTR_EVENT                   ; No, return this pointer event.
        WLDAI   16.,0                           ; Yes, the next pointer event
        XWADD   0,CURR_EVENT                    ;  now becomes the current
        XWSTA   0,CURR_EVENT                    ;  pointer event.
        WBR     MOVE_EVENT                      ; Continue scanning for movement
                                                ;  events.
```

Figure A–23. Listing of Program GRAPHICS_SAMPLE.SR (continued)

```
RTN_KEYS_EVENT:
; Return keystroke event.
        XWLDA   2,?OAC2,3               ; Load pointer to event packet.
        WLDAI   70,0                    ; Return keystroke event type in
        XWSTA   0,EVENT.TYPE,2          ;  event packet.
        XWLDA   0,CURR_EVENT            ; Load pointer to first
                                        ;  keystroke and store it in
        XWSTA   0,EVENT.CHAR_PTR,2      ;  event packet.
        XWLDA   0,CNT,3                 ; Load keystroke count, and
        XWSTA   0,EVENT.CHAR_CNT,2      ;  store it in event packet.
        XWLDA   0,CURR_EVENT            ; Load current event pointer.
        XWADD   0,CNT,3                 ; Add keystroke count, and
        XWSTA   0,CURR_EVENT            ;  store new current event.
        WBR     GET_GOOD_RTN            ; Return keystroke event.
RTN_PTR_EVENT:
; Return pointer event.
        XWLDA   2,?OAC2,3               ; Load pointer to event packet.
        XWLDA   0,CURR_EVENT            ; Load current event.
        WADDI   2,0                     ; Advance byte pointer to event
                                        ;  type.
        WLDB    0,1                     ; Load byte.
        XWSTA   1,EVENT.TYPE,2          ; Store event type in packet.
        WINC    0,0                     ; Advance byte pointer to
                                        ;  button.
        WLDB    0,1                     ; Load byte.
        XWSTA   1,EVENT.BTN,2           ; Store button number in packet.
        WINC    0,0                     ; Advance pointer to x-coord.
        WSUB    1,1                     ; Clear AC1.
        XJSR    TRANSLATE_LOC           ; Translate coordinate value.
        WBR     GET_ERROR_RTN           ; Handle an error.
        XWSTA   1,EVENT.LOC_X,2         ; Store x-coord. in packet.
        WADDI   6.,0                    ; Advance pointer to y-coord.
        WSUB    1,1                     ; Clear AC1.
        XJSR    TRANSLATE_LOC           ; Translate coordinate value.
        WBR     GET_ERROR_RTN           ; Handle an error.
        XWSTA   1,EVENT.LOC_Y,2         ; Store y-coord. in packet.
        XWLDA   0,CURR_EVENT            ; Load current event pointer.
        WADDI   16.,0                   ; Add pointer event length,
        XWSTA   0,CURR_EVENT            ;  and store new current event.
        WBR     GET_GOOD_RTN            ; Return pointer event.
GET_ERROR_RTN:
        XWLDA   0,?OAC0,3               ; Load error into caller's AC0.
        WBR     GET_RTN                 ; Return.
GET_GOOD_RTN:
        XWADI   1,?ORTN,3               ; Increment PC for good return.
GET_RTN:
        WRTN                            ; Return to caller.
```

Figure A–23. Listing of Program GRAPHICS_SAMPLE.SR (continued)

```
; TRANSLATE_LOC -- Translates a pointer coordinate into a real coordinate.
; On input, AC0 contains a byte pointer to the 6-byte ASCII sequence
;  which represents a pointer coordinate.  On output, AC1 contains the
;  translated 32-bit coordinate value.
; Temporary stack variables
I =             2                       ; Loop counter.
LOC =           4                       ; Temporary coordinate value.
TRANSLATE_LOC:
        WSSVR   2                       ; Save return block on stack.
        LDAFP   3                       ; Load frame pointer into AC3.
        WSUB    0,0                     ; Set temporary coordinate
        XWSTA   0,LOC,3                 ;  value to zero.
        XWLDA   1,?OAC0,3               ; Load byte pointer to ASCII
                                        ;  sequence from caller's AC0.
        WLDAI   30.,2                   ; Initialize bits-to-shift-left.
        WLDAI   1,0                     ; Initialize loop counter to
        XWSTA   0,I,3                   ;  one, and store on stack.
LOOP:
        XWLDA   0,I,3                   ; Load loop counter.
        WSLEI   6.,0                    ; End of loop?
        WBR     LOOP_END                ; Yes, break out of loop.
        WLDB    1,0                     ; Load current byte.
        WADDI   -60,0                   ; Subtract <060>.
        WANDI   77,0                    ; Get six least-significant
                                        ;  bits.
        WLSH    2,0                     ; Shift left appropriate amount.
        XWADD   0,LOC,3                 ; Accumulate coordinate value in
        XWSTA   0,LOC,3                 ;  temporary location on stack.
        WADDI   -6.,2                   ; Update bits-to-shift-left.
        WINC    1,1                     ; Advance byte pointer.
        XWADI   1,I,3                   ; Increment loop counter.
        WBR     LOOP                    ; Continue at top of loop.
LOOP_END:
        XWLDA   0,LOC,3                 ; Load coordinate value, and
        XWSTA   0,?OAC1,3               ;  store into caller's AC1.
        XWADI   1,?ORTN,3               ; Increment PC for good return.
        WRTN                            ; Return to caller.
```

Figure A–23. Listing of Program GRAPHICS_SAMPLE.SR (continued)

```
; Set up a place to save the window ID.
WIND_ID:
        0
FORM_ID:
        0
CHANNEL:
        .WORD 0
; Set up storage for some ?WIN_CREATE parameters.
WIND_WIDTH:
        512.
WIND_HEIGHT:
        512.
PIXEL_DEPTH =   2
BUFFER_SZ =     511.
PMAP:   .TXT    "@CONSOLE"
PMAP_PATH_LENGTH = 9.
WINDOW_NAME:
        .TXT "NEW_WINDOW"
NAME_LEN =      11.
WINDOW_TITLE:
        .TXT "GRAPHICS"
TITLE_LEN =     9.
WINDOW_PATH:
        .BLK 128.
; Define some masks for the ?WIN_CREATE flags word:
NEW_GRP =       1S?WIN_CRE.FLAGS.NEW_GRP
VT_SZ_X =       1S?WIN_CRE.FLAGS.VT_SZ_X
VT_SZ_Y =       1S?WIN_CRE.FLAGS.VT_SZ_Y
PIX =           1S?WIN_CRE.FLAGS.PIXEL_DEPTH
BUF_SZ =        1S?WIN_CRE.FLAGS.INPUT_BUF_SZ
UHIDE =         1S?WIN_CRE.FLAGS.UNHIDE_WINDOW
VOY =           1S?WIN_CRE.FLAGS.VIEW_ORG_X
VOX =           1S?WIN_CRE.FLAGS.VIEW_ORG_Y
; Allocate space for packets.
WIN_PKT:
        .BLK   ?WIN_PKT_LEN
WIN_CRE_SUBPKT:
        .BLK    ?WIN_CRE_LEN_REV1
GRAPH_PKT:
        .BLK    ?GRAPH_PKT_LEN
GRAPH_OPEN_SUBPKT:
        .BLK    ?GRAPH_OPEN_LEN
WRPAL_SUBPKT:
        .BLK    ?GRAPH_WRPAL_LEN
OPEN_PKT:
        .BLK    ?IOSZ
```

Figure A–23. Listing of Program GRAPHICS_SAMPLE.SR (continued)

```
; PALETTE ARRAY
PAL_ARRAY:      0
                0
                -1
                0
                -1
                -1
                0
                -1
                -1
                0
                0
                0
                0
                0
                -1
                0
; Allocate packet storage.
EVENT.TYPE =    0
EVENT.BTN =     EVENT.TYPE+2
EVENT.LOC_X =   EVENT.BTN+2
EVENT.LOC_Y =   EVENT.LOC_X+2
EVENT.CHAR_PTR = EVENT.LOC_Y+2
EVENT.CHAR_CNT = EVENT.CHAR_PTR+2
EVENT_PKT_LEN = EVENT.CHAR_CNT+2
EVENT_PKT:
        .BLK    EVENT_PKT_LEN
PTR_MOV =       1S?PTRDEV_SET_EVTS.EVTS.MOVEMENT
PTR_BTN_DWN =   1S?PTRDEV_SET_EVTS.EVTS.BTN_DOWN
PTR_BTN_UP =    1S?PTRDEV_SET_EVTS.EVTS.BTN_UP
PTR_WIN =       1S?PTRDEV_SET_EVTS.EVTS.WINDOW
PTR_DBL =       1S?PTRDEV_SET_EVTS.EVTS.DBL_CLICK
PTR_BTN1 =      1S?PTRDEV_SET_EVTS.BTNS.ONE
PTR_BTN2 =      1S?PTRDEV_SET_EVTS.BTNS.TWO
PTR_BTN3 =      1S?PTRDEV_SET_EVTS.BTNS.THREE
PTRDEV_PKT:
        .BLK    ?PTRDEV_PKT_LEN
PTRDEV_EVTS_PKT:
        .BLK   ?PTRDEV_SET_EVTS_LEN
X1 =            0
Y1 =            X1+2
X2 =            Y1+2
Y2 =            X2+2
LINE_PKT:
        .BLK    8.
ULC_X =         0
ULC_Y =         ULC_X+2
EXTENT_X =      ULC_Y+2
EXTENT_Y =      EXTENT_X+2
RFLOOD_PKT:
        .BLK    8.
; Allocate buffer.
CURR_EVENT:     0
EOB:            0
BUFFER_SZ =     511.    ; bytes
BUFFER:
        .BLK    (BUFFER_SZ+1)/2
        .END    CREATE_GRAPHICS
```

Figure A–23. Listing of Program GRAPHICS_SAMPLE.SR (concluded)

You must assemble GRAPHICS_SAMPLE.SR with a MASM–created symbol table file (.PS file) that includes long names for the offsets in the parameter packets of the three system calls ?GRAPHICS, ?PTRDEVICE, and ?WINDOW. This symbol table file must also include GIS II instructions. Each release of AOS/VS software includes such a file; its default pathname is :UTIL:MASM_32CHAR.PS.

Also, you must execute GRAPHICS_SAMPLE.PR on a graphics terminal with a pointer device that is part of the DS/7xxx family of computers. If you don't have this hardware, GRAPHICS_SAMPLE.PR will terminate with an *Illegal function for device* error message.

The assembly command is

```
) XEQ MASM/PS=MASM_32CHAR.PS GRAPHICS_SAMPLE ⤶
```

The Link command is

```
) XEQ LINK/STACK=1024 GRAPHICS_SAMPLE ⤶
```

The command to execute the program is

```
) XEQ GRAPHICS_SAMPLE ⤶
```

When the program begins execution you will see a blue rectangle with the word "GRAPHICS" in a gray bar at the top of the rectangle. Draw lines by clicking button 1, moving the pointer device, and clicking button 1 again. Draw bars by clicking button 2, moving the pointer device, and clicking button 2 again. Click button 2 twice to end the program.

The following events lead to the result that Figure A–22 contains. The events begin after the CLI command

```
) XEQ GRAPHICS_SAMPLE ⤶
```

1. Move the pointer device so that the arrow is in the left center region of the blue rectangle.
2. Click button 1.
3. Move the pointer device in a 4 o'clock direction to draw a gray line about 4 inches long.
4. Click button 1.
5. Move the pointer device so that the arrow is about 1.25 inches below and slightly to the left of the location you selected in step 1.
6. Click button 1.
7. Move the pointer device in a 2 o'clock direction to draw a gray line about 3 inches long.
8. Click button 1.
9. Move the pointer device so that the arrow is about 0.25 inch below and 0.5 inch to the left of the location you selected in step 1.
10. Click button 2.
11. Move the pointer device to the right to draw a thin yellow bar about 4.25 inches long.

12. Move the pointer device down about 1.5 inches to expand the thin yellow bar into a thick yellow bar.

13. Click button 2.

14. Now you see results close to those in Figure A–22.

15. Double–click button 2 to end the program's execution and to return to the CLI.

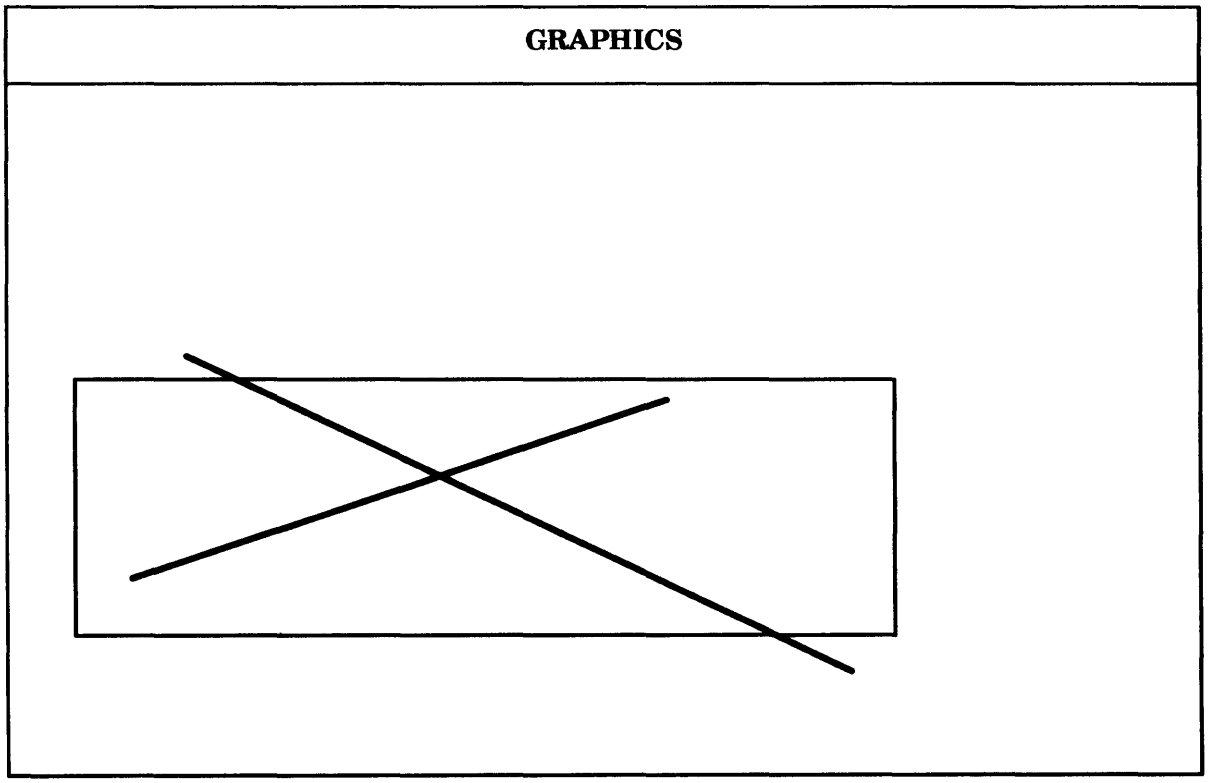


Figure A–24. Possible Results of Executing Program GRAPHICS_SAMPLE

End of Appendix

Appendix B
System Log Record Format

This appendix 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.

Each record in the SYSLOG file contains an *event code* that identifies the event that it is logging. The event code can be either one of the standard AOS/VS event codes, or a special event code that AOS/VS allows you to define within your programs.

To write a record into the SYSLOG file, a process must issue the ?LOGEV system call; to issue the ?LOGEV call, the process must have Superuser mode turned on. You should also examine the system call ?SYLOG to understand how to control all system, Superuser, and CON0 logging functions.

This appendix lists the standard operating system event codes and what they represent. It also describes the format of the records that log these events.

Reporting the Contents of the SYSLOG File

You can report the contents of the SYSLOG file either by using the REPORT utility or by writing a program that reads and reports on the contents of the SYSLOG file.

If you use the REPORT utility, it will report on *only* those records that contain the standard operating system event codes. Since you define the meaning of any special event codes, *and* specify the contents of the records that describe those events, REPORT cannot report on the events you log with those records.

If you have defined special event codes and you want to read and report on the records that contain them, or if you want to report on only certain standard AOS/VS events, you must write a program to do so.

Reading the SYSLOG File

To read the SYSLOG file, you should open it for dynamic reads. You can declare an integer*2 array for the record header using a 0 base (e.g., in FORTRAN 77, INTEGER*2 HEADER(0:7)). This allows you to use the 0–base subscripts that we show. You can also declare an integer*2 array for each different record length, less 8 elements. At runtime, you can read the header array, check the length from words 0 and 1. Then, if the record has information other than the header, you can read the remainder of the record into the array of its length. Then, in the header array, you can check word 5 for the message code. If this is a code you want, you can break down the header and array, and format them for output, and then read the next record. If you don't care about the code, you can simply read the next record.

You can get the record length from the two–word length descriptor by — in FORTRAN 77 — equivalencing element 0 of the descriptor to a 4–byte integer.

Most record formats have a specific length. Formats for events like process creations and file opens, however, don't have a specific length. Instead, their length varies with the length of the username and/or pathname stored in them. These flexible format records are recorded only when logging was turned on with /DETAIL = FULL. The flexible record lengths help conserve disk space (since otherwise every record would have to be as long as the longest one – and these events occur quite often). The only records with flexible length are

- Process events. These include process creation (code 910), chaining to another process (917), and loading a program into a ring (ring load, ?RINGLD system call, 916).

- File access events. Events with flexible–length records are file create (code 929); file open (code 920); file delete (code 924), file rename (codes 938 and 942); initialize LDU (code 937); release LDU (code 928); change file ACL (codes 939 and 943); read user data area, UDA (code 925); and write UDA (code 926).

In each of these flexible length records – as in all records – the record length appears in the first two 16–bit words in each record.

Record Header Format

Figure B–1 shows the header that begins each log record. The sixth 16–bit word in the header is the event code. In this and the following figures, all subscript/offsets and event codes are decimal. In the records themselves, all numeric values are octal.

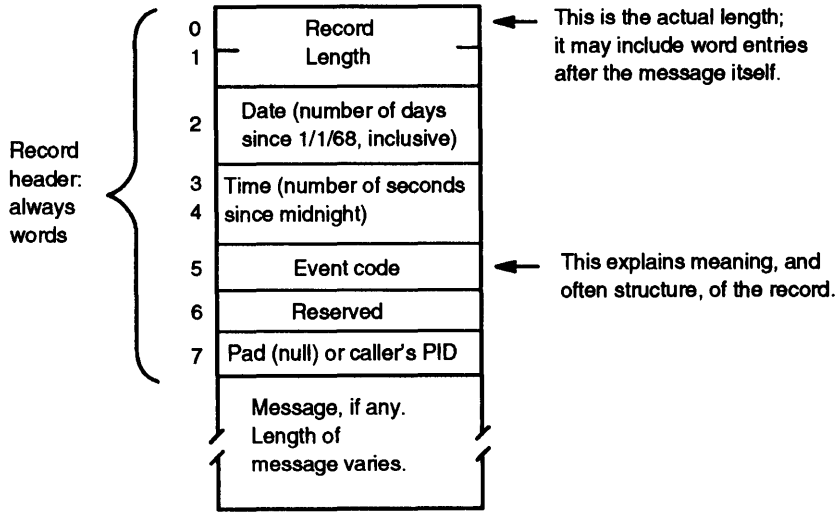


Figure B–1. Log Record Header

SYSLOG Record Formats

The SYSLOG file stores the following standard record types. AOS/VS writes these records into the SYSLOG file; however, AOS/VS writes those record types for which we note "Error Log" into the system's error log file, with the exception of Events 1, 2, 8, and 9 which it writes into both files.

Table B–1 summarizes all record types by event code, including their record length. Records with length noted as 0 consist only of the header.

Figure B–2 describes records that are longer than the header. The SYSLOG file stores numeric values in octal and ASCII characters as ASCII. Where padding is needed, SYSLOG uses nulls (ASCII 000). The symbol # means "number."

Event codes 900–999 and 1200–1299 are logged only when system logging was turned on with the /DETAIL = FULL switch. *Do not use codes 900–999 and 1200–1299 for your special event codes.* The following list details the remaining code catagories:

1 — 39	Administrative
40 — 899	AOS/VS Hardware Error Codes
1000 — 1023	AOS/VS Hardware Error Codes
1024 — 1199	AOS/VS DG User (Exec/CLI) Administrative
1300 — 2047	DG User

Table B–1. SYSLOG Event Codes and Record Lengths

Event Code (decimal)	Meaning	Message Length (beyond headers)
0	Unused	
1	SYSLOG logging turned on.	4
2	SYSLOG logging turned off.	8 through 1016 (varies).
3	Process termination.	26
4	Device error — Error log.	32
5	Pad record (padding only).	8 through 148 (varies).
6	Unused by MV/Family processors.	3
7	Power failure — Error log.	3
8	AOS/VS and log file revision (follows logging started record, 1).	4

(continued)

Table B–1. SYSLOG Event Codes and Record Lengths (cont.)

Event Code (decimal)	Meaning	Message Length (beyond headers)
9	Change of Day/Time.	7
10	Reserved.	—
11	System Inconsistency.	7
12	MV 40000 HA Quiesce restart (warm repair).	3
13	MV 40000 HA JP Retry	3
14	Manipulate exclusion bitmap for selective logging.	256
15	Manipulate exclusion bitmap for Superuser logging.	64
16	Superuser logging started/stopped event.	4
40	Power restored — Error log.	0
41	Fatal AOS/VS error — Error log.	33
42	AOS/VS hang — Error log.	0
43	Single-bit ERCC error, MV/4000 — Error log.	5
44	Multibit ERCC errors, MV/4000 — Error log.	5
45	ERCC Sniff 1/0 errors, MV/4000 — Error log.	5
73	SCP reset (not recorded).	0
74	SCP request completed (not recorded).	0
75	Host-SCP error (not recorded).	0
76	Host-SCP buffer full error — Error log.	0
77	SCP time-out — Error log.	0
78	SCP interface degrade — Error log.	0
79	SCP-request-to-host error — Error log.	0
80	SCP buffer not cleared — Error log.	0

(continued)

Table B–1. SYSLOG Event Codes and Record Lengths (cont.)

Event Code (decimal)	Meaning	Message Length (beyond headers)
81	Host-request-to-SCP error — Error log.	0
96	SCP logging enabled — Error log.	0
97	SCP logging disabled — Error log.	0
98	Main processor halt — Error log.	0
99	BOOT issued (MV/8000) — Error log.	0
100	Power failure — Error log.	0
101	Power restore — Error log.	0
102	Air flow fault — Error log.	0
103	Overtemp fault (not recorded).	0
104	Transfer to battery backup — Error log.	0
105	Reserved.	—
106	ERCC error, MV/8000 and MV/6000 — Error log.	4
107	Microsequencer parity error — Error log.	0
108	System cache parity error — Error log.	0
109	Cache to Bank controller parity error — Error log.	0
110	IOC bus parity error — Error log.	0
111	S-bus timeout — Error log.	0
112	S-bus parity error — Error log.	0
113	Operating system error (unused).	0
114	Diskette log error (MV/8000).	0
115	Infinite protection fault — Error log.	0
116	Infinite page fault — Error log.	0
117	Instruction cache enabled — Error log.	0

(continued)

Table B–1. SYSLOG Event Codes and Record Lengths (cont.)

Event Code (decimal)	Meaning	Message Length (beyond headers)
118	Instruction cache disabled — Error log.	0
119	Reserved.	—
120	Reserved.	—
121	System reset (not recorded).	0
122	ATU accelerator enabled — Error log.	0
123	ATU accelerator disabled — Error log.	0
125	XEQ DTOS command (MV/8000).	0
126	Bad return from DTOS (MV/8000).	0
127	HALT command (MV/8000) — Error log.	0
128	CONTINUE command (MV/8000) — Error log.	0
129	START command (MV/8000) — Error log.	0
130	INIT command (MV/8000) — Error log.	0
131	Bank controller ERCC report disable — Error log.	0
132	Good return from DTOS (MV/8000).	0
133	Hard interrupt (not recorded).	0
141	Unsolicited error from peripheral (H.A.D.A./MV)	3
142	Soft tape error	17
901	Reserved.	0
910	Process created.	Varies
911	Reserved.	0
912	Process terminated by superior process.	4

(continued)

Table B–1. SYSLOG Event Codes and Record Lengths (cont.)

Event Code (decimal)	Meaning	Message Length (beyond headers)
913	Superuser turned on or off.	Varies
914	Superprocess turned on or off.	Varies
915	Access devices turned on or off (?IDEF turns on; ?IRMV turns off).	4
916	Process loaded program into ring (used ?RINGLD system call).	Varies

(continued)

Table B–1. SYSLOG Event Codes and Record Lengths (cont.)

Event Code (decimal)	Meaning	Message Length (beyond headers)
917	Process chained to another process.	Varies
918	PMGR assigned a console to a process.	5
919	PMGR revoked a process's console assignment.	5
920	File opened.	Varies
921	Reserved.	—
922	File closed.	4
923	Reserved.	—
924	File deleted, access by pathname. See also error code 931).	Varies
925	File user data area (UDA) read, access by pathname. See also 932.	Varies
926	File UDA written, access by pathname. See also 933.	Varies
927	File UDA created, access by pathname. See also error code 934.	Varies
928	Logical disk unit (LDU) released, access by pathname.	Varies
929	File created, access by pathname.	Varies
930	Reserved.	—
931	File deleted, access by channel number.	4
932	File user data area (UDA) read, access by channel number.	4
933	File UDA written, access by channel number.	4
934	File UDA created, access by channel number.	4
935	Reserved.	—
936	Reserved.	—

(continued)

Table B–1. SYSLOG Event Codes and Record Lengths (cont.)

Event Code (decimal)	Meaning	Message Length (beyond headers)
937	Logical disk unit (LDU) initialized.	Varies
938	File renamed, access by pathname. See also error code 942).	Varies
939	File ACL changed, access by pathname. See also error code 943).	Varies
940	Reserved.	—
941	Reserved.	—
942	File renamed, access by channel number.	Varies
943	File ACL changed, access by channel number.	Varies
944	Reserved.	—
945	Shared file opened, first open.	Varies
946	Shared file opened, subsequent open.	7
947	Permit access to protected file call (used to modify shared file) occurred.	9
948	Job processor initialized (?JPINIT system call issued).	4
949	Job processor released (?JPREL system call issued).	4
950	Job processor moved to another logical processor (?JPMO system call issued).	4
951	Reserved.	—
952	Logical processor created (?LPCREA system call issued).	4
953	Logical processor deleted (?LPDELE system call issued).	4
954	Reserved.	—
955	Class assignments requested or set (?LPCLASS system call issued).	4

(continued)

Table B–1. SYSLOG Event Codes and Record Lengths (cont.)

Event Code (decimal)	Meaning	Message Length (beyond headers)
956	Class IDs requested or set (?CLASS system call issued).	4
957	Class scheduling matrix requested or set (?CMATRIX system call issued).	4
958	Reserved.	4
959	Class scheduling enabled or disabled (?CLSCHED system call issued).	4
960	Process's user locality changed (?LOCALITY system call issued).	4
961	?MIRROR system call issued successfully.	64
962	?MIRROR system call issued unsuccessfully.	64
963	LDU mirror image released by system.	64
964	System Manager privilege (SYSMGR) turned on or off.	Varies
965	Process protection trap.	6
966	Window create.	43
967	Window marked for deletion.	43
968	Block a Process (?BLKPR system call).	4
969	Establish a Connection (?CON system call).	4
970	Break a Connection (?DCON ?DRCON system calls).	4
971	Change process type (?CTYPE system call).	5
972	Remap address space (?LMAP system call).	7
973	Map to a memory mapped device (?MAPDY system call).	4
974	Become a system operator (?OPERON, ?OPEROFF system calls).	4
975	Change system ID (?SSID system call).	4
976	Log the secondary error.	40

(continued)

Table B–1. SYSLOG Event Codes and Record Lengths (cont.)

Event Code (decimal)	Meaning	Message Length (beyond headers)
977	Reserved.	—
978	Move bytes to/from user (?MBTU, ?MBFU system calls).	4
979	Send an IPC (?ISEND system call).	10
980	Send an IPC and post ?IREC (?IS.R).	10
981	Reserved.	—
982	LDU mirror established (hardware synchronized).	64
983	LDU mirror established (hardware unsynchronized).	64
984	LDU mirror established (software synchronized).	64
985	LDU mirror established (software unsynchronized).	64
986	User changed their group list.	10
1023	UPSC power supply fault.	25
1024	Console connect time.	26
1025	Unit mount time.	26
1026	Privileged user logon.	9
1027	Pages printed.	25
1028	Reserved.	—
1029	Reserved.	—
1030	RMA accounting.	17
1031	XODIAC RMA Error.	11
1064	FTA accounting.	26
1065	General event (LOGEVENT).	Varies
1066	DG/SNA accounting.	22
1067	X.25 Accounting.	44
1068	X.25 Error.	14

(continued)

Table B–1. SYSLOG Event Codes and Record Lengths (cont.)

Event Code (decimal)	Meaning	Message Length (beyond headers)
1069	Message Transfer Service (MTA) accounting.	47
1200	Reserved.	—
1201	Reserved.	—
1211	Labeled medium (tape) mounted.	29
1212	Labeled medium (tape) dismounted.	29
1213	User logon.	27
1214	File printed.	155
1215	Invalid logon attempt.	26
1220	User profile created.	18
1221	User profile deleted.	18
1222	User profile renamed.	34
1223	User profile opened (e.g., by EXEC when a user starts to log on).	18
1224	User profile read (follows event code 1223).	20
1225	User profile written to (follows event 1223).	20
1226	User profile closed (follows event 1223).	18
1227	Reserved.	—
1228	Reserved.	—
1229	LOCK_CLI locked or unlocked.	4

(concluded)

NOTE: Codes 900–999 and 1200–1299 are logged only if logging was enabled with /DETAIL= FULL.

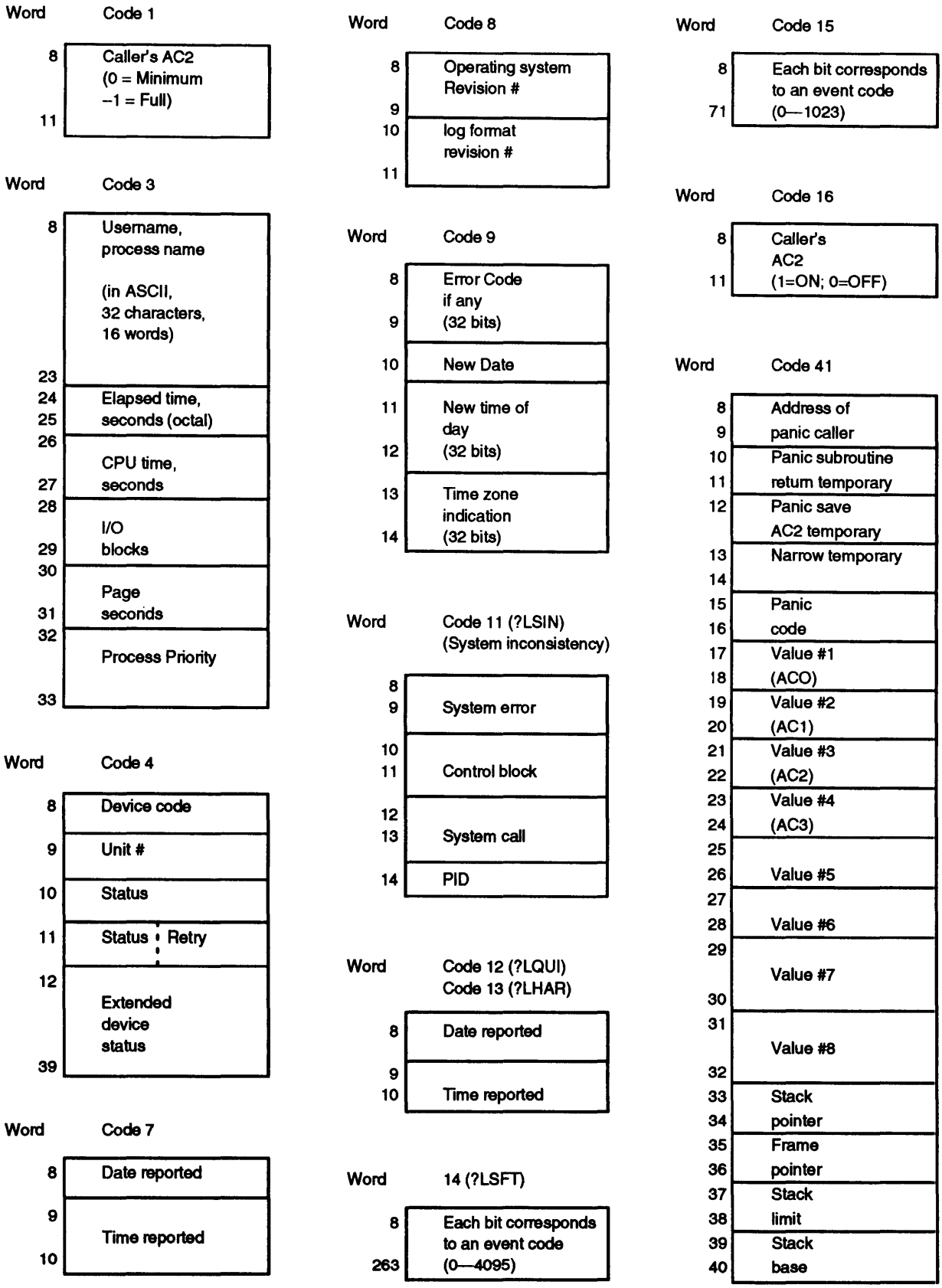


Figure B–2. Log Record Codes, Events, and Message Lengths, Excluding Header (continues)

Word	Code 43, 44, or 45
8	Code (43, 44, or 45)
9	Cause (CPU, I/O access, sniff, other)
10	Physical page #1
11	Double word on module
12	Syndrome bits

Word	Code 106
8	ERCC code
9	Module
10	Plane #
11	Bit #

Word	Code 141
8	DIA Status
9	DIB Status
10	Device Code

Word	Code 142
8	Device code
9	Unit number
10	Suppress Soft error flag (0=suppress, 1=report)
11	Total number of re–writes
12	Total number of write errors corrected
14	Total number of bytes written to the media

(continued)

Word	Code 142
16	Total number of re–reads
17	Total number of read errors corrected
19	Total number of bytes read from the media
21	Acceptable Error/Bytes ratio limit
22	Marginal Error/Bytes ratio limit
23	Error/Bytes trans–ferred ratio
24	Acceptance levels: Good = 01 Marginal = 02 Bad = 03

Word	Code 910 (process created)
8 9	Error code if any (Note 1)
10	0
11	Privilege bits (?PROC format)
12	0
13	New PID
14 21	Username, in ASCII, 16 characters, 8 words
22 n	Program (.PR) file pathname (ASCII) or terminated by null

Word	Code 912 (process terminated)
8 9	Error code if any (Note 1)
10 11	Target PID

Word	Code 913, 914, 964
8 9	Error code (if any)
10 11	n (0 means no change, 1 means off, 2 means on)
12 19	ASCII USERNAME (Note 4)

Word	Code 915
8 9	Error code if any (Note 1)
10 11	n (0 means no change, 1 means off, 2 means on)

Word	Code 916 (?RINGLD)
8 9	Error code if any (Note 1)
10	0
11	Ring number
12 n	Program (.PR) file pathname loaded, in ASCII, terminated by null

Word	Code 917 (chain)
8 9	Eff or code if any (Note 1)
10 n	Program (.PR) file pathname, in ASCII, terminated by null

Figure B–2. Log Record Codes, Events, and Message Lengths, Excluding Header (continued)

Codes 918, 919 (Assign/revoke assignment of terminal by PMGR)

Word	
8 9	Error Code, if any.
10	PID of process receiving/losing terminal assignment.
11	Device code of IAC or system console.
12	Secondary code (terminal line #).

Code 920 (file opened)

Word	
8 9	Error code if any (Note 1)
10	Flag word (Note 2)
11	Channel number
12 n	Pathname, in ASCII, terminated by null

Codes 922, 931, 932, 933, and 934 (file accessed by channel number)

Word	
8 9	Error code if any (Note 1)
10	Flag word (Note 2)
11	Channel number

Codes 924, 925, 926, 927, 928, 929 (file deleted, UDA accessed, LDU released, file created)

Word	
8 9	Error code if any (Note 1)
10	Flag word (Note 2)
11	Pathname, in ASCII, terminated by null (for code 927, pathname is the
n	LDU name)

Code 937 (LDU initialized)

Word	
8 9	Error code if any (Note 1)
10	Flag word (Note 2)
11	Disk unit name(s) in LDU (e.g., @DPJ10), in ASCII, separated by null (if more than one), ends
n	with two nulls.
n + 1 n + m	LDU name, in ASCII, terminated by null

Code 938 (file renamed)

Word	
8 9	Error code if any (Note 1)
10	Flag word (Note 2)
11	New filename, in ASCII, terminated by null
n + 1 n + m	Original pathname, in ASCII, terminated by null

Code 939 (file ACL changed)

Word	
8 9	Error code if any (Note 1)
10	Flag word (Note 2)
11	New ACL in call ?SACL format: user, null, value, user, null, value, and so on (user is the username, in ASCII; value is the access value, explained in Note 3); a null terminates the ACL.
n +1 n+m	Pathname, in ASCII, terminated by null.

Code 942 (file renamed by channel #)

Word	
8 9	Error code if any (Note 1)
10	Flag word (Note 2)
11	Channel number
12 n	New filename, in ASCII, terminated by null

Code 943 (file ACL changed by channel #)

Word	
8 9	Error cods if any (Note 1)
10	Flag word (Note 2)
11	Channel number
12 n	New ACL in call ?SACL format, as described in 939 above; ends with a null.

Figure B–2. Log Record Codes, Events, and Message Lengths, Excluding Header (continued)

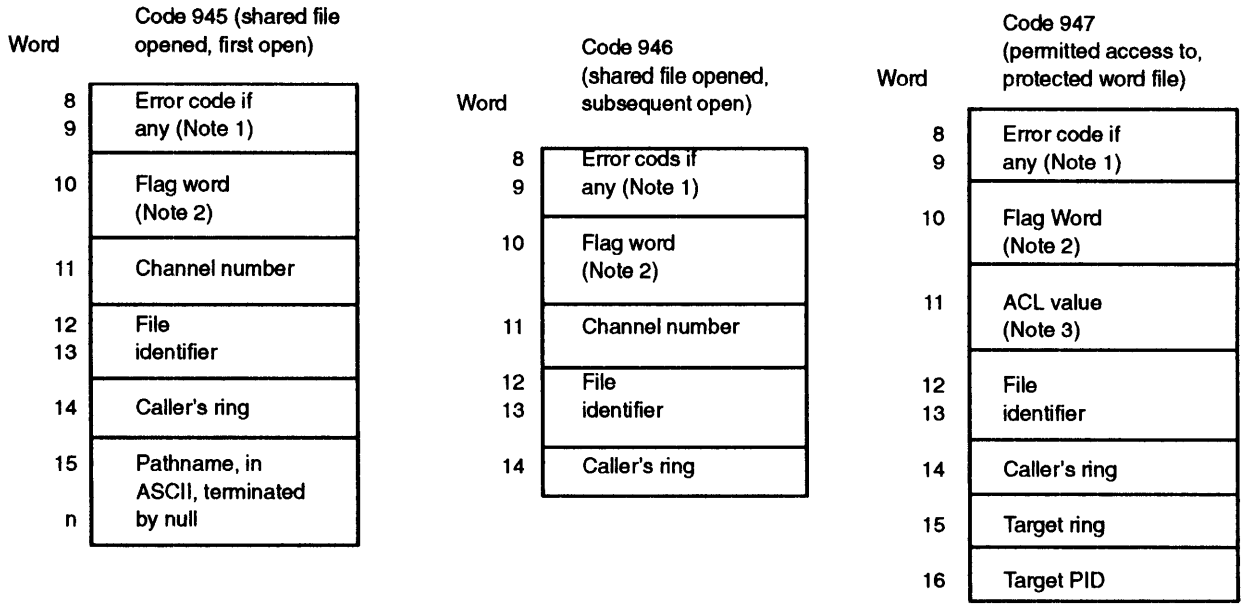


Figure B–2. Log Record Codes, Events, and Message Lengths, Excluding Header (continued)

Codes 948 through 960

Word	
8	Error code, if any.
9	
10	Parameter value
11	

Code 961—963, 982—985

Word	
8	ASCII LDU name followed by image names (separated by nulls)
71	

Code 965

Word	
8	Error Code (Trap Type)
9	
10	PC of instruction causing the trap
11	
12	Address triggering this trap (if any)
13	

Code 966, 967

Word	
8	Error code if any
9	
10	Requester's PID
11	ASCII path name of window (terminated by a null)
50	

Code 968

Word	
8	Error code if any (Note 1)
9	
10	Target PID
11	

Code 969

Word	
8	Caller's ring
9	Server's PID
10	
11	Server's ring

Code 970

Word	
8	Caller's Ring
9	Target PID
10	
11	Target Ring

Code 971

Word	
8	Error code if any (32 bits)
9	
10	Pid of changing process
11	
12	New Type

Code 972

Word	
8	Error code if any (32 bits)
9	
10	Mapped ring
11	Logical source page address
12	
13	Physical source page address
14	

Code 973

Word	
8	MAP IN/OUT
9	Target device
10	Physical address of 1st page
11	

Code 974

Word	
8	Global daemon name ID, or 0
9	
10	n (1 means off, 2 means on)
11	

Code 975

Word	
8	Error Code
9	
10	New System ID
11	

Code 976 (?LERR)

Word	
8	Error code
9	
10	Return address
11	
12	Return address
13	.
	.
	.
47	

Code 978

Word	
8	Target PID (32 bits)
9	
10	n (1 means to; 2 means from)
11	

Code 979, 980

Word	
8	Error code if any (32 bits)
9	
10	Sender's Pid (32 bits)
11	
12	Sender's ring
13	Sender's LPN
14	Receiver's Pid (32 bits)
15	
16	R's Ring
17	R's LPN

Code 982—985 (SEE 961—963)

Figure B–2. Log Record Codes, Events, and Message Lengths, Excluding Header (continued)

Word Code 986

8	Error code
9	
10	Double null terminated ASCII group or username list
17	

Word Code 1023

8	UPSC power supply fault code

Word Code 1024 or 1025

8	Username (in ASCII, 16 characters,
15	8 words)
16	Device name (in ASCII, 32 characters,
31	16 words)
32	Minutes used
33	

Word Code 1026

8	Username (in ASCII, 16 characters,
15	8 words)
16	Privilege bits (?PROC format)

Word Code 1027

8	Username (in ASCII, 16 characters,
15	8 words)
16	Device name (in ASCII, 32 characters,
31	16 words)
32	Pages printed

Word Code 1030

8	Remote host ID
9	Virtual Circuit #1
10	AOS/VS error code
11	Username (in ASCII, 16 characters,
18	8 words)
19	
20	Not used
21	
22	Connect time
23	
24	Request count

Word Code 1031

8	Remote host
9	Virtual Circuit #1
10	Termination code
11	Username . . .
18	

Word Code 1064

8	Remote host ID
9	Virtual Circuit #
10	AOS/VS error code
11	Username (in ASCII, 16 characters,
18	8 words)
19	
20	Not used
21	
22	Connect time
23	Service type
24	
25	I/O block count
26	
27	# of packets sent
28	
29	# of packets received
30	
31	# of bytes sent
32	
33	# of bytes received

Word Code 1065

8	Message (in ASCII, padded to multiple of 8, max is 48 characters,
31	24 words)

Figure B–2. Log Record Codes, Events, and Message Lengths, Excluding Header (continued)

Word Code 1066

8	Logical unit (LU) address
9	
10	Time LU closed
11	
12	Time LU opened
13	# of request
14	units received
15	
16	# of bytes received
17	# of Request
18	units sent
19	
20	# of bytes sent
21	SNA customer name (username: processname, in ASCII. 18 characters,
29	9 words)

Word Code 1067

8	Linkname (in ASCII, 16 characters,
15	8 words)
16	Channel #
17	Virtual Circuit
18	Connect type
19	Reason code
20	Username: processname (in ASCII, 32 characters,
35	16 words)
36	Connection address (in ASCII, 15 characters,
43	8 words)
44	Connect time
45	
46	# of packets sent
47	# of packets received
48	# of bytes sent
49	
50	# of bytes received
51	

Word Code 1068

8	Linkname (in ASCII, 16 characters,
15	8 words)
16	Channel #
17	Virtual Circuit
18	Flag word
19	XODIAC error code
20	Diagnostic error code
21	AOS/VS error code

Word Code 1069 (MTA Accounting)

8	Remote host ID
9	Virtual Circuit ID
10	Error Code
11	Username (16 characters)
.	.
.	.
.	.
18	
19	Reserved
20	
21	Connect time
22	
23	Service type
24	I/O block count
25	
26	# of packets sent
27	
28	# of packets received
29	
30	# of bytes sent
31	
32	# of bytes received
33	
34	Reserved (14 words)
.	.
.	.
.	.
47	

Word Codes 1211 and 1212 (labeled medium mounted and dismounted)

8	Error code if
9	any (Note 1)
10	Username (in ASCII, 16 characters,
17	8 words)
18	Device name (in ASCII, 32 characters,
33	16 words)
34	Volume ID (volid), 6 characters,
36	3 words

Word Code 1213 (user log on)

8	Error code if
9	any (Note 1)
10	Username (in ASCII, 16 characters,
17	8 words)
18	Console name (in ASCII, 32 characters,
33	16 words)
34	Privileges (?PROFILE call format)

Figure B–2. Log Record Codes, Events, and Message Lengths, Excluding Header (continued)

Code 1214 (file printed)

Word	
8 9	Error code if any (Note 1)
10 17	Username (in ASCII, 16 characters, 8 words)
18 33	Device name (in ASCII, 32 characters, 16 words)
34	of pages printed
35 162	Pathname of file (in ASCII, 256 characters, 128 words)

Code 1215 (invalid log on attempt)

Word	
8 9	Error code if any (Note 1)
10 17	Username (in ASCII, 16 characters, 8 words)
18 33	Console name (in ASCII, 32 characters, 16 words)

Codes 1220, 1221, 1223, and 1226 (user profile created, deleted, opened, and closed)

Word	
8 9	Error code if any (Note 1)
10 25	Profile name (in ASCII, 32 characters, 16 words; terminated by null)

Code 1222 (user profile renamed)

Word	
8 9	Error code if any (Note 1)
10 25	Original profile name (in ASCII, 32 characters, 1 6 words; terminated by null)
26 41	New profile name (in ASCII, 32 characters, 16 words)

Codes 1224 and 1225 (user profile, field read or written)

Word	
8 9	Error code if any (Note 1)
10 25	Profile name (in ASCII, 32 characters, 16 words; terminated by null)
26 27	Field number in description packet (?profile call format)

Code 1229 (LOCK_CLI locked or unlocked)

Word	
8 9	Error code if any (Note 1)
10 11	n (0 means no change, 1 means unlocked, 2 means locked.)

Figure B–2. Log Record Codes, Events, and Message Lengths, Excluding Header (concluded)

Note 1: If an error occurred on this event, the AOS/VS error code is stored, right–justified, in these two 16–bit words (this 32–bit double word). For example, the value 242 (octal) in this double word means the user's access was rejected an an *Execute access denied* error message. If no error occurred on the event, the value of the double word is 0.

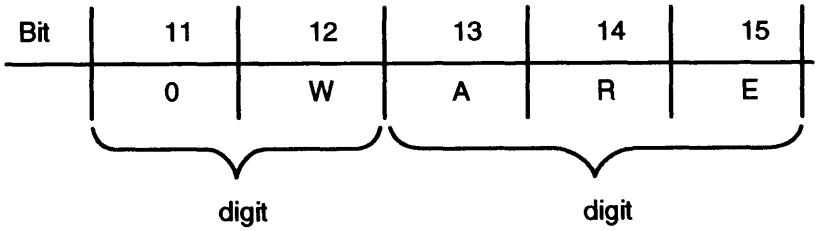
Note 2: The flag word normally contains 0, unless something extraordinary happened to prevent the complete record from being written (for example, a hard error on disk). the error code field within the record will probably not show an error, since it records things like user access privilege errors. The flag word codes (numeric values) are

040000 Pathname field not available. This means a problem (like a hard error on disk, also shown in ERROR_LOG) prevented the entire pathname from being recorded. it can also occur on an attempt to initialize an LDU if the user lacked E access to the PER entry for the unit.

020000 This means the first variable field was incomplete (perhaps) for the same reasons as flag code 40000). The first variable field occurs only in records with codes 937, 938, 939, 942, and 943 (LDU initialize, file rename, and ACL change).

010000 This means that one or more fixed fields (like channel number) were not available. It can happen on ?OPEN or ?SOPPF if the pathname resolution fails.

Note 3: The file ACL value is a composite of the value of bit settings. The structure of log records longer than 8 words is as follows:



Thus OWARE access has the value 37, WARE 17, ARE 7, RE 3, and E 1.

Note 4: The USERNAME is only present when the PRIVILEGE is turned ON.

Anatomy of a System Log File Record

An easy way to examine a system log file is to use the DISPLAY utility. You can apply the /DECIMAL switch to get a decimal display. NO TAG shows both an octal and decimal display of a logfile. You can get similar displays via X DISPLAY :SYSLOG and X DISPLAY/DECIMAL :SYSLOG.

```
    0  000000 000010 011635 000000 101443 000001 000000 000000 .......#........
   10  000000 000040 011635 000000 101475 000003 000000 000000 .......=........
   20  047520 035060 030063 000000 000000 000000 000000 000000 OP:003..........
   30  000000 000000 000000 000000 000000 000000 000000 000000 ................
   40  000000 000002 000000 000320 000000 000011 000000 000012 ................

  400  000000 000050 011635 000000 102146 002001 000000 000000 ...(.....f......
  410  045101 041513 000000 000000 000000 000000 000000 000000 JACK............
  420  046524 041060 000000 000000 000000 000000 000000 000000 MTBO............
  430  000000 000000 000000 000000 000000 000000 000000 000000 ................
  440  000003 000000 000000 000000 000000 000000 000000 000001 ................

 1410  000000 000020 011635 000000 120060 002051 000000 000000 .........0.).....
 1420  051165 067156 064556 063440 051145 073040 031056 030060 Running Rev 2.00
 1430  000000 000020 011635 000000 120102 002051 000000 000000 .........B.).....
 1440  046117 043505 053105 047124 020114 064566 062563 020441 LOGEVENT Lives!.


    0  0      8      5021   0      33571  1      0      0      .........#......
    8  0      32     5021   0      33597  3      0      0      ........=.......
   16  20304  14896  12339  0      0      0      0      0      OP:003..........
   24  0      0      0      0      0      0      0      0      ................
   32  0      2      0      208    0      9      0      10     ................
    .
  256  0      40     5021   0      33894  1025   0      0      ...(.....f.......
  264  19009  17227  0      0      0      0      0      0      JACK............
  272  19796  16944  0      0      0      0      0      0      MTBO............
  280  0      0      0      0      0      0      0      0      ................
  288  3      0      0      0      0      0      0      1      ................
    .
  776  0      16     5021   0      41008  1065   0      0      .........0.).....
  784  21109  28270  26990  26400  21093  30240  12846  12336  Running Rev 2.00.
  792  0      16     5021   0      41026  1065   0      0      .........B.).....
  800  19535  18245  22085  20052  8268   26998  25971  8481   LOGEVENT Lives!!.
    .
```

Figure B–3. An Octal and Decimal DISPLAY of a System Log File

The first record starts with number 0 in both the octal and decimal display. The second word in this record appears as 000010 in the octal display, 8 in the decimal display. This word tells the length. For this record it is 10 octal (8) words. The code, in the fifth word of the record, is 1. From Figure B–2, you can tell that code 1 means that system logging was on.

Figure B–4 shows a breakdown of the first record in octal and decimal.

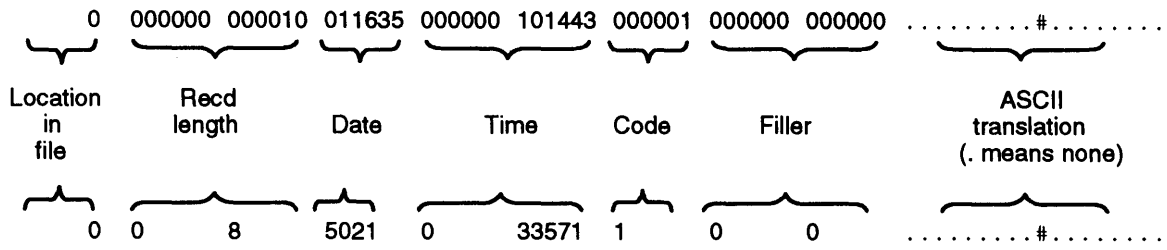


Figure B–4. Octal and Decimal Versions of a SYSLOG Record

The second record starts at location 10 octal (8). It is 40 octal (32) words long. The code in the fifth word is 3, which means that the record is a process termination message. The ASCII appears in the field off to the right (this field is not in the record, but is a convenience supplied by DISPLAY). Looking at the fields of this record, you can see (in the last line) the elapsed time, processor time, I/O blocks, and page–seconds. The third record shown appears later on in the file. It has a record length of 50 octal (40). The fifth word, 2001 octal (1025), describes it as a unit mount. Again, you can see the ASCII off to the right. JACK used the mounted tape for 3 minutes.

The fourth and fifth records appear still later in the file. Each is 24 octal (16) words long. The fifth word in each header is 2051 (1065). This means it is an event record, written by the LOGEVENT command. Once more, you can see the ASCII on the right. The report that the REPORT program would generate from this log file would have a lot of columnar information, and its numbers would be decimal – as shown in the earlier REPORT figures.

End of Appendix

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

086–000196 updates
093–000543

C

D

086–000196 updates
093–000543

E

086–000196 updates
093–000543

F

G

H

I

086–000196 updates
093–000543

J

K

L

086–000196 updates
093–000543

M

N

O

P

Q

R

086–000196 updates
093–000543

S

086–000196 updates
093–000543

T

U

V

W

X

Y

Z

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 Shipping 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

☐ 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

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 ?R Through ?Z

093–000543–02

Cut here and insert in binder spine pocket